Table of Contents

Cloud Computing......1

AWS Cloud......2
- HECC AWS Cloud: Overview......2
- AWS Cloud File Transfer Overview......6
- AWS Dynamic Front End......9
- Building and Running Applications in the AWS Cloud......13
- Managing AWS PBS Jobs Launched From a PFE......15
- Managing PBS Jobs Launched from an AWS Dynamic Front End......18
- Staging Between Pleiades and AWS within PBS Jobs Submitted from a PFE......20
- Transferring Files Between a Pleiades or AWS Front End and AWS S3 Storage......22
- Transferring Files Between AWS S3 Storage and Your AWS PBS_O_WORKDIR Directory......23
- Creating AWS Job-Time Filesystems......25
- Transferring Files Between AWS S3 and Job-Time Filesystems......28
- Optional Job-Time Filesystem Related Directives......30
- Examples of Job-Time Filesystem Related Directives......32
- Order of Operation of Cloud Directives......35
- Cloud Billing Units and Accounting......37
- AWS PBS Resources and Examples......39
- Using Jupyter Notebook or JupyterLab in the AWS Cloud......41
- Running a Singularity Container Image in AWS......44
- AWS Cloud FAQs......45
  - Can I get more information about your AWS Public Cloud offering?......45
  - On a high level, what kinds of work are supported?......45
  - On a high level, what kinds of work are not supported?......45
  - What CPU and GPU node types are supported in AWS?......45
  - Is there a way to apply for free access?......46
  - We are looking for a cloud environment with software installed. Do you provide Software as a Service (SaaS)?......46
  - What operating system is used in your cloud?......46
  - How much does it cost?......46
  - Who do I send my NASA Work Breakdown Structure (WBS) funding?......46
  - I do not currently have a NAS account. I can bring funding. How do I start?......46
  - After my cloud environment is created, how do I log in?......46
  - Do you allow non-NASA projects to use your cloud offering?......46
  - How do you plan to allow non-NAS users to login to your cloud?......47
  - Will I be charged if the cloud resources have problems or do not perform as I expected?......47
  - Who do I contact if I need help with issues or job failures on AWS?......47

AWS Cloud ML/AI......48
- AWS Cloud: Machine Learning/Artificial Intelligence......48
- Accessing the AWS Management Console Through Cloudtamer......49
- Obtaining Short-Term AWS Credentials (Access Keys)......50
- Creating a New Front End in AWS......51
- Creating and Accessing AWS SageMaker Notebook Instances......53
- Creating a Sagemaker Studio IDE......54
- Using a Jupyter Environment Outside of SageMaker......55

Containers......56
- Singularity......56
  - Enabling User Defined Software Stacks with Singularity......56
  - Building an Image Using the Singularity Build Tool on Pleiades or Your Local Host......58
  - Running a Singularity Container Image on Pleiades......60
  - Converting Docker Images to Singularity for Use on Pleiades......64
  - Running Singularity in Hybrid MPI Mode......66

Table of Contents

Containers

Best Practices for Running Singularity on NAS Systems..70
Running a Singularity Container Image in AWS..72

Cloud Computing

AWS Cloud

HECC AWS Cloud: Overview

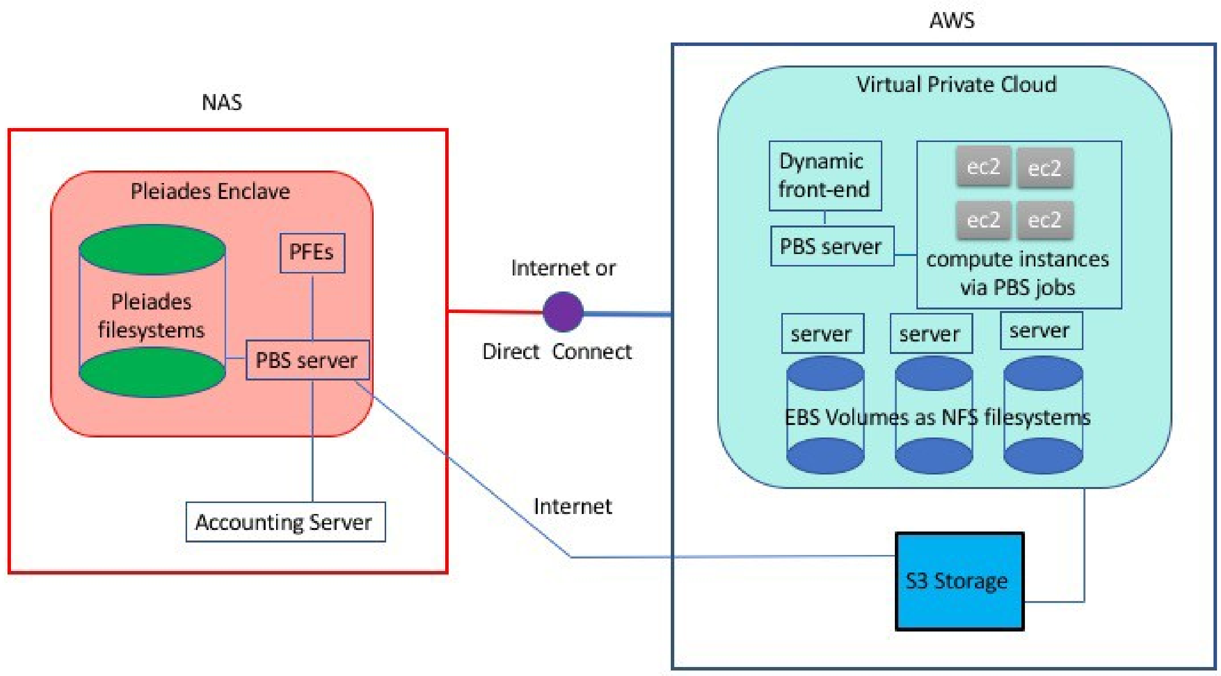
Starting summer 2019, we now offer the Amazon Web Services (AWS) Cloud on a pay-for-use basis. If you are a principal investigator (PI) directly affiliated with NASA, and you are interested in paying with NASA funding to use AWS Cloud for your projects, please contact us at support@nas.nasa.gov. Access to the AWS Cloud for users in a PI's project will be granted after funding is received and the AWS environment is configured for the PI's project.

Webinar: In addition to the articles in this section, you can find more information in our user training webinar, "Overview of HECC Pay-for-Use AWS Cloud." The recording and presentation slides are available in the HECC webinars archive.

This article provides a high-level overview of the HECC AWS Cloud. Follow the links in each section to learn more details about using the AWS Cloud.

HECC AWS Cloud Environment

As shown in this diagram, a project's HECC AWS Cloud environment includes both the HECC resources located at the NAS facility, and HECC's AWS resources. The resources at NAS are shared among projects, while those at AWS are private for each project. Some of the resources are optional, depending on the need of a project.



NAS-Located Resources

The purpose and functions of the NAS-located resources are described in this section.

Pleiades Front-End Systems (PFEs)

Each user in a project must have a NAS account in order to log in to a PFE. Access to AWS resources is accomplished via an SSH session from a PFE. Authentication to AWS is done behind the scenes.

From a PFE, you can remotely manage:

- PBS jobs submitted to AWS
- Data stored in AWS S3 storage

NAS PBS Server

A PBS server located at NAS, currently called clpbs-01, is used for accepting and checking PBS jobs submitted to run on AWS.

Pleiades /nobackup Filesystems

To run a job from NAS, you must submit it to AWS from your Pleiades /nobackup filesystem. The PBS output/error files of the AWS batch jobs are sent back to the $PBS_O_WORKDIR on your /nobackup filesystem.

Your /nobackup filesystem is also one of the sources or destinations for different types of file transfers between NAS and AWS.

Accounting Server

The accounting server manages your cloud allocations in Cloud Billing Units (CBUs). You can check your cloud allocation and usage by running the NAS **`acct_ytd`** and **`acct_query`** tools.

AWS Resources

The purpose and functions of the AWS resources are described in this section.

Note: The term *instance* used at AWS is logically equivalent to the term *node* used at NAS.

Dynamic Front End

In order to save costs, there is no static front end for AWS that runs 24x7. Instead, you launch a dynamic front end when you need one, and then shut it down when you're done. You only pay for the time that the dynamic front end is running.

A dynamic front end uses one AWS Elastic Compute Cloud (EC2) instance of your choice.

Use the dynamic front end to compile applications, and/or to manage and check PBS jobs or data on AWS. A limited number of software modules are available in the **`/nasa`** directory; if you need additional software, you must install it yourself under your own directory.

For more information, see AWS Elastic Compute Cloud (EC2) instance types.

AWS PBS Server

The PBS server at AWS coordinates with the PBS server at NAS to manage:

- Batch jobs submitted remotely from a PFE
- Batch jobs submitted locally from a dynamic front end

Note: Be sure to read AWS PBS Resources and Examples before submitting a batch job to AWS.

To save costs, the AWS PBS server is shut down when there are no batch jobs to manage.

The hostname of the AWS PBS server is different for each project. To find yours, use the following script on a PFE:

```
/u/scicon/tools/bin/aws_pbs_host
```

If you have multiple AWS projects, each with its own PBS server, add **`--group`** ***`gid`*** to the **`aws_pbs_host`** command to find the PBS server associated with that GID.

The instance type used for the server is currently m5.xlarge. It may change in the future.

Compute Instances for Batch Jobs

AWS offers many types of EC2 instances that you can use to run batch jobs. The default regions are **US West** for public cloud and **AWS GovCloud (US)** for government cloud. The HECC AWS Cloud uses the Linux AMI operating system.

Pricing varies with regions and operating systems.

For more information, see:

- EC2 instance types
- EC2 pricing
- EC2 regions and availability zones

Filesystem Servers

A filesystem requires a server with sufficiently high bandwidth to the filesystem. A c5.18x instance (similar to an Electra Skylake node) is typically chosen for the persistent filesystem described below. For a job-time filesystem, the instance chosen varies depending on the size to be allocated to the filesystem. For example:

- A c5d.18x instance for sizes below 1.8 terabyte (TB)
- An h1.16x instance for sizes above 4 TB and below 16 TB
- A c5.18x instance with Elastic Block Store (EBS) for all other sizes

To save costs, the server is shut down when the filesystem is not in use. If you need multiple filesystems, then you will need multiple servers.

Electric Block Store (EBS) Volumes as Filesystems

You can use Electric Block Store (EBS) Volumes as filesystems. AWS offers several EBS volume types and pricing. Depending on your project's need, different volumes can be provisioned into two types of filesystems:

- A persistent filesystem is independent of a batch job. Data in the filesystem persists even when the filesystem is not used. You pay for the persistent cost of the size provisioned, and for the time when the filesystem server is running.

  This type of filesystem is usually configured by HECC staff upon your request at the beginning of your project.
- A job-time filesystem is created at the beginning of a batch job and terminated at the end of the job. The data in this filesystem is lost at the end of the job. You pay for the size provisioned for the lifetime of the job, and the uptime of the filesystem server (if needed). Some job-time filesystems might not use EBS volumes.

For more information, see:

- EBS volumes
- EBS volume types
- EBS pricing

S3 (Simple Storage Service) for Long-Term Storage

Among the multiple S3 (Simple Storage Service) classes offered by AWS, HECC uses the standard class. Pricing depends on space used, the frequency of request, and the amount of data transferred out of S3 to the Internet. For more information, see:

- S3 storage classes
- S3 pricing

AWS Cloud File Transfer Overview

This article provides a high-level view of available methods for file transfers among various NAS and Amazon Web Services (AWS) filesystems or storage systems.

Transferring Files Between NAS and AWS Systems

Transfers Between Your PFE /home or /nobackup Filesystem and an AWS Persistent Filesystem

To use these methods, you must have a configured persistent filesystem and the IP address of a live AWS dynamic front end.

Option 1: Initiate Transfer from a PFE by Using scp

To transfer your file:

```
pfe% scp -i ~/.ssh/id_rsa_yours iii.jjj.kkk.lll:/nobackup/your_user_name/filename .
pfe% scp -i ~/.ssh/id_rsa_yours -r src iii.jjj.kkk.lll:/nobackup/your_user_name
```

Replace **`id_rsa_yours`** with the name of the RSA private key file that is set up for you to connect to AWS under your Pleiades **`.ssh`** directory, and replace **`iii.jjj.kkk.lll`** with the actual IP address of your live AWS dynamic front end.

Option 2: Initiate Transfer from the AWS Dynamic Front End by Using sup + shiftc

Before you can use this method, follow the instructions in Downloading SUP to Enable Remote Transfers to install **`sup`** into your AWS **`~/bin`** directory.

To transfer your file:

```
aws% which sup
~/bin/sup
aws% sup shiftc pfe21.nas.nasa.gov:~/file1 .
```

`sup + shiftc` will choose a transfer protocol for you based on what's available in your AWS environment. You can also specify a transfer protocol explicitly, for example, **`scp`**, as follows:

```
aws% sup scp file2 pfe21.nas.nasa.gov:
```

Note: You will need to provide your NAS password and PIN+passcode to generate the SUP authentication key, which is valid for 7 days.

Option 3: Initiate Transfer from the AWS Dynamic Front End by Using scp

Before you can use this method, you must set up SSH passthrough to allow one-step login from AWS to a PFE.

Note: You will need to modify the NAS-provided config file in order to get it working on the HECC

AWS Cloud. Modify the file by replacing **`Host *.nas.nasa.gov`** with **`Host pfe*.nas.nasa.gov lfe*.nas.nasa.gov`**.

To transfer your file:

```
aws% scp file2 pfe21.nas.nasa.gov:
```

Transfers Between Your PFE /home or /nobackup Filesystem and AWS S3

All users have space under their project's S3 storage. You can use the following set of **`nas_s3_`*`xxx`*** commands on a PFE to view, transfer, and delete files and directories under your S3 root location:

- View: **`nas_s3_ls`**
- Transfer: **`nas_s3_put`** and **`nas_s3_get`**
- Delete: **`nas_s3_del`**

Read this article for more information.

Transfers Between your PFE /nobackup Filesystem and an AWS Filesystem Accessible from a PBS Job

Through a PBS job submitted from your Pleiades /nobackup directory, you can add directives to stagein or stageout files or directories to/from the job's $PBS_O_WORKDIR on AWS. The $PBS_O_WORKDIR could be on a persistent filesystem or a job-time filesystem, depending on the set up of the PBS job.

The available directives are:

```
#CLOUD -stagein_file=path_to_file/file
#CLOUD -stagein_dir=path_to_dir
#CLOUD -stageout_file=path_to_file/file
#CLOUD -stageout_dir=path_to_dir
#CLOUD -stageout_file_delete=output
```

Read this article for more information.

Transferring Files within AWS

Transfers Between S3 and the Dynamic Front End

You can use the set of **`nas_s3_`*`xxx`*** commands on an AWS dynamic front end to view, transfer, and delete files and directories under your S3 root location:

- View: **`nas_s3_ls`**
- Transfer: **`nas_s3_put`** and **`nas_s3_get`**
- Delete: **`nas_s3_del`**

Read this article to find more information.

Transfers Between S3 and $PBS_O_WORKDIR within a PBS Job

You can use the following directives to copy files between your $PBS_O_WORKDIR on AWS and your S3 storage on AWS.

Note: $PBS_O_WORKDIR could physically be on a persistent or a job-time filesystem on AWS.

```
#CLOUD -get_file=path_to_file/file
#CLOUD -get_file=sub_dir/file:upper_dir
#CLOUD -get_dir=path_to_dir
#CLOUD -get_dir=sub_dir:upper_dir
#CLOUD -put_file=src:optional_dest_dir
#CLOUD -put_dir=src_dir:optional_dest_dir
```

Read this article for more information.

Transfers Between S3 and Job-Time Filesystems within a PBS job

You can use these directives to copy files between a job-time filesystem and your S3 storage on AWS.

```
#CLOUD -volume_get=s3_folder_to_copy_data_from
#ClOUD -volume_put=s3_folder_to_save_data_to
```

Read this article for more information.

AWS Dynamic Front End

Because there is no static front end for AWS, you launch a dynamic front end only when you need one, and then shut it down when you're done. You only pay for the time that the front end is running.

This article provides instructions for launching, accessing, and terminating an AWS dynamic front end from a Pleiades front end (PFE), along with some basic usage information.

Launching a Dynamic Front End

To launch a dynamic front end, run the **`aws_fe`** command from a PFE to start an instance, as shown below. This command creates a batch job that runs behind the scenes to request the resources. There is no limit on the number of AWS front ends you can request or run.

```
pfe% /u/scicon/tools/bin/aws_fe -h
aws_fe  (optional resource specs)
 will start a front end for the user in AWS that will run for a default 1 hour and have 1 core
       --group X      Charge against the given group. (Your default cloud group is scicon)
       --region X     Launch in the given region if the project supports multiple regions
       --container X  Enable the given container technology enabled. Current options: singularity
       --jupyter      Create a front end instance with a jupyter notebook running.
       --jupyterlab   Create a front end instance with a jupyter lab running.
-t X  (--hour X)      Request X hours of walltime
-c X  (--cores X)     Create instance with at least X cores
-m X  (--mem X)       Create instance with at least X G of ram
-g X  (--gpu X)       Create instance with the specified GPU type attached. Options: v100/a100/k80
-n X  (--ngpu X)      Request X cards of the specified type in the instance
-a X  (--arch X)      Request the 'X' model of instances, not valid with -g/--gpu.
                           Options: m4/m5/c4/c5/c5a
-l    (--list)        Print login instructions for the existing front end instance the user has,
                           should one exist
-k    (--kill)        Terminate an existing front end instance. If only 1 FE exists it will be killed.
                           If multiple FEs exist it returns an error unless -j is also used.
-j X  (--job X)       When used with --kill will kill the FE with the given ID.
                           See output from --list for the value of ID for each front end.
-d    (--dcv)         Create a front end with DCV running, instructions on logging in will be emailed
      (--shared)      Allows all users in the same group to be able to use the DCV gui as well
-b X  (--band X)      Request X amount of network bandwidth per instance. Options: 1/10/25 (Gbps)
                           Note that using this option with others can result in an impossible
                           combination of options.
                           The request will fail silently
```

The following PDFs list the CPU and GPU node types that are supported in AWS:

- CPU node types (PDF)
- GPU node types (PDF)

Recommendation: For your convenience, include **`/u/scicon/tools/bin`** in your PATH so that **`aws_fe`** can be found without including the full path in the command line. You can also accomplish this by loading a module on a PFE:

```
module load /u/scicon/tools/modulefiles/scicon_cli_tools
```

If you have more than one cloud project group ID (GID), one of them will be used as a default for **`aws_fe`** as listed in the output of **`aws_fe -h`**. Use the **`--group`** option to specify a non-default cloud GID. If you want to change your default cloud GID, send an email to support@nas.nasa.gov to open a ticket.

Running the **`aws_fe`** command without any arguments will launch a front end consisting of 1 node, with at least 8 GB of memory and 1 CPU core, that will remain for 1 hour.

Note: The current default instance type assigned is m5.xlarge, which has 2 cores and 16 GB, and costs $0.192/hour in the US West public cloud. The default may be different for your project.

Example: Launch an Instance with a K80 GPU

This example launches an instance that has at least one K80 GPU on it:

```
pfe% aws_fe -l
No front end to display

pfe% aws_fe -t 4 -g k80 -n 1
The front end is booting, when the instance is ready an email
will be sent to your NAS email with login instructions.
```

Sample Email Confirmation

If your launch is successful, you will receive an email similar to the following to confirm that the instance has started and provide login instructions:

`Your front end instance has started in AWS.`
`To log in use the following from a PFE machine:`
`env SSH_AUTH_SOCK="" ssh -i ~/.ssh/id_rsa_yours` ***`your_nas_username@iii.jjj.kkk.lll`***

Note: If you don't see an email a few minutes after launching, check your email spam folder. This is likely to happen for users who forward their NAS emails to (or through) Gmail.

TIP: If your PBS server at AWS is asleep, your first launch attempt might fail. If this happens, you will receive an email with the message "Job rejected by all possible destinations." This first failed attempt will wake up the PBS server; you can then repeat the **`aws_fe`** command to try launching the instance again.

Accessing a Dynamic Front End

To access a dynamic front end, run SSH from a PFE, as shown below.

Note: The first time you access the assigned front end, you might be prompted with the question: **`Are you sure you want to continue connecting (yes/no)?`**. Type **`yes.`**

```
pfe% env SSH_AUTH_SOCK="" ssh -i ~/.ssh/id_rsa_yours your_nas_username@iii.jjj.kkk.lll
The authenticity of host 'iii.jjj.kkk.lll (iii.jjj.kkk.lll)' can't be established.
ECDSA key fingerprint is SHA256:oE1wd9e/vGwo4cPj6oHwF182II1jo2p3H8gaSXBXGoY.
Are you sure you want to continue connecting (yes/no)? yes
-bash-4.2$
```

where ***`id_rsa_yours`*** is the one that was set up under your Pleiades **`.ssh`** directory for connecting to AWS, and ***`iii.jjj.kkk.lll`*** is the IP address of your live dynamic front end.

The front end host key is regenerated each time a new front end is started. This invalidates the copy in your Pleiades **`~/.ssh/known_hosts`** file, which has the IP address used by a previous front-end node or compute node. When this happens, you will see a message similar to the following:

```
pfe% env SSH_AUTH_SOCK="" ssh -i ~/.ssh/id_rsa_yours your_nas_username@iii.jjj.kkk.lll
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
```

```
The fingerprint for the ECDSA key sent by the remote host is
SHA256:4QGzjDB+xiYq3liOyC5Ymbmw4GbuMblcKKdwrV0rjYE.
Please contact your system administrator.
Add correct host key in /homeX/username/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /homeX/username/.ssh/known_hosts:4
ECDSA host key for iii.jjj.kkk.lll has changed and you have requested strict checking.
Host key verification failed.
```

If you see this message, you must modify your **`.ssh/known_hosts`** file on the PFE to remove the host key entry with the IP number from your new **`aws_fe`** request, and retry the SSH connection.

Once you are in, be aware that:

- The default shell is bash.

  ```
  [username@awsfe-XXXXX ~] echo $0
  -bash
  ```

  To change to csh for the current shell, do:

  ```
  [username@awsfe-XXXXX ~] csh
  [username@awsfe-XXXXX ~] echo $0
  csh
  ```

  To change to csh permanently on all AWS machines, send an email to support@nas.nasa.gov with your username and the GID(s) used on your AWS cloud environment.
- There is only a minimal .bashrc file set up for you. You should create your own startup scripts or modify the minimal .bashrc file to set variables such as $PATH, or to load certain modules if your job relies on loading them inside a startup file.
- There is a limited number of software modules installed under the **`/nasa`** directory. Use the command **`module avail`** to see what is available. If you need a program that is not already available, you will have to install under your own directory.

Terminating the Dynamic Front End

When you are done using the dynamic front end, you must terminate it. If you simply exit the front end, it is still owned by youâ   and you will continue to pay for itâ   until either it reaches the wall-time limit specified at start time, or you terminate it by using one of the following methods:

- Use the **`-k`** option of **`aws_fe`** command (recommended method):

  ```
  pfe% aws_fe -k [-j X]
  ```

  Note: Include **`-j X`** if you have more than one front end running. You can find *X* from the output of **`aws_fe -l`**.
- Delete it using the **`aws_qdel`** command described in this article.

After you terminate the front end, you will receive an email with the front end usage statistics. (Check your spam folder if you do not see one.) Here is an example:

```
____________________________________________________________________
Job Resource Usage Summary for 3563.clpbs-01.nas.nasa.gov
    Total Runtime            : 00:59:46
    Job Startup Time         : 00:02:04
    Time Spent In PBS Script : 00:57:41
    Walltime Requested       : 01:00:00
    Charged To               : cstaff
    Job Finished             : Wed Jul 3 14:37:29 2019
    Instance Types (ondemand): 1 m5.xlarge
```

```
    EBS Usage                : 8577331200 bytes
    NAS overhead charge      : 0.000 percent
    Job Costs                : $0.192358499543
```

Building and Running Applications in the AWS Cloud

This article provides basic information on building and running applications in the HECC AWS Cloud.

Due to differences in the operating systems and software stacks, an executable built on Pleiades will not run in the AWS Cloud. For information on transferring source files from Pleiades before building your executable in your AWS Cloud environment, see AWS Cloud: File Transfer Overview.

Available Modules

This list shows the default set of software packages installed under the **/nasa** directory in your AWS environment:

- Intel compiler
- PGI compiler
- Intel MPI
- OpenMPI
- CUDA

You (or NAS staff) can also install Python from the Intel or Conda distributions, if needed.

If you need any other software packages, you must install them into a non-root-privileged filesystem yourself. If you need a licensed software package, you must provide the license on your own.

Building and Running MPI Applications

The HPE MPT library is not available in your AWS Cloud environment. Use either Intel MPI or OpenMPI instead.

WARNING: The variable **StrictHostKeyChecking** is set to **no** in a system level configuration file, since both Intel MPI and OpenMPI rely on running SSH before launching an MPI process. If you reset this variable to **yes** or **ask** in your .ssh/config file, your MPI job will stall.

Using Intel MPI

Load an Intel MPI module in your build and run sessions. For example:

```
aws% module load intel-mpi/2019.3.062
```

To compile, load either an Intel compiler or a PGI compiler. This example uses an Intel compiler:

```
aws% module load comp-intel/2019.3.062
aws% mpiifort your_program.f
aws% mpiicc your_program.c
```

If you use a PGI compiler, replace **mpiifort** with **mpif90** and **mpiicc** with **mpicc**.

To run, use **mpiexec** in an interactive session or a PBS batch script:

```
mpiexec -np xx ./a.out
```

Using OpenMPI

In your system startup script, such as .bashrc or .profile (for bash) and .cshrc (for csh), include loading of an OpenMPI module. For example:

```
module load openmpi/4.0.1
```

This is needed if you plan to run OpenMPI executables across nodes.

To build an executable, use either an Intel compiler or a PGI compiler. This example uses a PGI compiler:

```
aws% module load comp-pgi/19.04
aws% mpif90 your_program.f
aws% mpicc your_program.c
```

To run, use **`mpiexec`** in an interactive session or a PBS batch script:

```
mpiexec -np xx ./a.out
```

Managing AWS PBS Jobs Launched From a PFE

This article provides information on managing your Amazon Web Services (AWS) PBS jobs submitted from a Pleiades front end (PFE).

The PBS server at NAS for managing AWS cloud job is clpbs-01. It coordinates with another PBS server at AWS to get your job burst into and running on AWS, and sending the PBS output/error file back to Pleiades.

Note: The **`aws_pbs_host`**, **`aws_qstat`**, and **`aws_qdel`** scripts described in this article are located under the directory **`/u/scicon/tools/bin`** on Pleiades. The instructions below assume that you have included **`/u/scicon/tools/bin`** in your $PATH.

Finding Available Queues

To find available queues, type the command:

```
pfe% qstat -q @clpbs-01
```

Among the queues listed in the output, you only have direct access to the *cloud* queue. The other queues, such as *frontend* and *s3op*, are used by NAS-controlled scripts (*aws_fe* and *nas_s3_xxx*) for managing the AWS dynamic front end and data at AWS S3 on your behalf.

The limits of each queue are configured in the PBS server at AWS. To see them, use:

```
pfe% aws_qstat -Q @`aws_pbs_host`
pfe% aws_qstat -Q @`aws_pbs_host --group gid` (to check for a non-default GID)
```

Note: If you get the following message, your PBS server at AWS has likely gone to sleep, for cost saving purposes. Submitting a job or two should wake up your server.

```
Connection timed out
qstat: cannot connect to server your_aws_pbs_server.nas.nasa.gov (errno=110)
```

Submitting Jobs (qsub)

You must submit jobs to AWS from your Pleiades /nobackup filesystem.

From a PFE, you can submit regular batch jobs to the *cloud* queue, but not interactive jobs, using the following command:

```
pfe% qsub -q cloud@clpbs-01 job_script
```

To submit jobs to a non-default AWS GID, add the**`-W group_list=gid`** option, where **`gid`** is in the form of **`zxxxx`** (i.e., the letter "z" followed by four numbered digits):

```
pfe% qsub -q cloud@clpbs-01 -W group_list=gid job_script
```

or add it inside your PBS script:

```
#PBS -W group_list=gid
```

Note: If you get the following message within a few minutes of submitting your job, your PBS server at AWS likely went to sleep, for cost saving purposes; however, by this time it should have been awakened and subsequent jobs should go through.

```
PBS Job Id: xxxx.clpbs-01.nas.nasa.gov
Job Name:   test.pbs
Aborted by PBS Server
Job rejected by all possible destinations
```

Checking Jobs (qstat or aws_qstat)

When a job is submitted from a PFE, it is under the control of the PBS server clpbs-01 at the beginning, and running **qstat** at that time on a PFE will show the job.

```
pfe% qstat -a @clpbs-01
```

The status of a job is H (held) while any files are copied to temporary storage on the AWS cloud. When the copy is completed, the status changes to M, indicating that it was moved to the PBS server at AWS. The job then disappears from clpbs-01. However, you can see past jobs using the following:

```
pfe% qstat -x @clpbs-01
```

After clpbs-01 hands over the control for the job to AWS, the job is visible from your AWS PBS server, using:

```
pfe% aws_qstat @`aws_pbs_host`
pfe% aws_qstat @`aws_pbs_host --group gid` (to check for a non-default GID)
```

You can get more detailed information about a job using the following:

```
pfe% aws_qstat -f jobid.clpbs-01.nas.nasa.gov@`aws_pbs_host --group gid`
```

When a job finishes, it may take a few minutes for the PBS output file to show up in your Pleiades directory. You can check whether your job has completed using:

```
pfe% aws_qstat -xqst @`aws_pbs_host --group gid`
```

Deleting Jobs (aws_qdel)

To delete jobs submitted to AWS from a PFE, use the script **aws_qdel**. Note that you cannot delete these jobs using the standard **qdel** command, because it involves the coordination of two PBS servers.

```
pfe% aws_qdel jobid
or
pfe% aws_qdel jobid.clpbs-01.nas.nasa.gov
```

To delete a job for a non-default AWS GID, use:

```
pfe% aws_qdel --group gid jobid
or
pfe% aws_qdel --group gid jobid.clpbs-01.nas.nasa.gov
```

Location of Your PBS Output/Error Files

Within a few minutes after your job has completed or been terminated, the PBS output/error files are sent back to your $PBS_O_WORKDIR (where you submitted the job from) on Pleiades.

AWS Job Accounting

AWS charges for the compute instances (CPU/GPU), filesystems, storage, and network for transferring data out of AWS. NAS charges an overhead cost, which is a percentage of the total AWS charge. The Job Costs entry includes both the AWS charge and the NAS overhead charge.

Example Job Accounting Summary

```
____________________________________________________________________
Job Resource Usage Summary for 3552.clpbs-01.nas.nasa.gov
    Total Runtime            : 00:03:03
    Job Stage In Time (free) : 00:00:00
    Job Startup Time         : 00:01:03
    Time Spent In PBS Script : 00:02:00
    Job Stage Out Time       : 00:00:00
    Walltime Requested       : 02:00:00
    Execution Queue          : AWS Cloud
    Charged To               : cstaff
    Job Finished             : Wed Jul  3 10:38:09 2019
    Instance Types (ondemand): 1 m5.xlarge
    EBS Usage                : 8577331200 bytes
    S3 Usage                 : 0 bytes
    Charged Bandwidth Usage  : 0 bytes
    NAS overhead charge      :  0.000 percent
    Job Costs                : $0.00981639861027
____________________________________________________________________
```

Managing PBS Jobs Launched from an AWS Dynamic Front End

This article provides information on managing your PBS jobs submitted from an AWS dynamic front end.

For additional information, see also, Managing AWS PBS Jobs From a PFE.

Note: The **`aws_pbs_host`**, **`aws_qstat`**, and **`aws_qdel`** scripts described in this article are located under **`/u/scicon/tools/bin`** on Pleiades. The instructions below assume that you have included **`/u/scicon/tools/bin`** in your $PATH.

After launching and using SSH to access an AWS dynamic front end, you can manage PBS jobs as follows.

Finding Available Queues

To find available queues and their limits, use the -q or -Q option:

```
aws% qstat -q
or
aws% qstat -Q
```

Among the queues listed in the output, you have direct access only to the *cloud* queue, which routes the job to the execution queue *cloud_exec*. The other queues, such as *frontend* and *s3op*, are used by NAS-controlled scripts (*aws_fe and nas_s3_xxx*) for managing the AWS dynamic front end and data at AWS S3 on your behalf.

Submitting Jobs (qsub)

You can submit both interactive and batch jobs to the 'cloud' queue, using:

```
aws% qsub -I -q cloud -lselect=1...
aws% qsub -q cloud job_script
```

Note: You should be able to tell by the hostname included in the prompt whether you are in the front end (*awsfe-XXXXX*) or in the interactive PBS session (*compute-XXXXX).*

Note: The wall-time requested for an interactive PBS session should not be longer than the remaining time of your AWS dynamic front-end session.

Checking Jobs (qstat or aws_qstat)

You can check the job status either by using **`qstat`** on your AWS front end or by using **`aws_qstat`** on a PFE, as follows:

- On the AWS Front-End

  ```
  aws% qstat -a
  aws% qstat -nu your_user_name
  ```
- On the PFE

  ```
  pfe% aws_qstat -a @`aws_pbs_host`
  pfe% aws_qstat -a @`aws_pbs_host --group gid`(to check a job for a non-default GID)
  pfe% aws_qstat -nu your_user_name@`aws_pbs_host --group gid`
  ```

Deleting Jobs (qdel or aws_qdel)

You can delete a job using **`qdel`** on your AWS front end or **`aws_qdel`** on a PFE.

- On the AWS Front-End

```
aws% qdel jobid
```

- On the PFE

```
pfe% aws_qdel jobid  (note, the jobid here is just the numerical part)
pfe% aws_qdel --group gid jobid (to delete a job for a non-default GID)
pfe% aws_qdel --group gid jobid.your_aws_pbs_server
pfe% aws_qdel --group gid jobid.your_aws_pbs_server.nas.nasa.gov
```

Finding Your PBS Output/Error Files

When a job exits, the PBS output file (which includes job accounting data) and error files are placed in the $PBS_O_WORKDIR on AWS, they are *not* sent back to Pleiades.

Handling Job Accounting

AWS charges for the compute instances (CPU/GPU), filesystems, storage, and network for transferring data out of AWS. NAS charges an overhead cost, which is a TBD percentage of the total AWS charge. The Job Costs entry includes both the AWS charge and the NAS overhead charge.

Example Job Accounting Summary

```
____________________________________________________________________
Job Resource Usage Summary for 3622.your_aws_pbs_server.nas.nasa.gov
    Total Runtime            : 00:04:10
    Job Stage In Time (free) : Submitted from AWS, no stage in
    Job Startup Time         : 00:02:05
    Time Spent In PBS Script : 00:02:03
    Job Stage Out Time       : 00:00:01
    Walltime Requested       : 00:05:00
    Execution Queue          : AWS Cloud
    Charged To               : scicon
    Job Finished             : Fri Apr 12 16:49:57 2019
    Instance Types (ondemand): 1 m5.2xlarge
    EBS Usage                : 16773959680 bytes
    S3 Usage                 : 0 bytes
    Charged Bandwidth Usage  : 0 bytes
    NAS overhead charge      :  0.000 percent
    Job Costs                : $0.0268173415057
____________________________________________________________________
```

Staging Between Pleiades and AWS within PBS Jobs Submitted from a PFE

This article describes how to use **`stagein`** and **`stageout`** directives in a PBS job script to copy files between Pleiades and the $PBS_O_WORKDIR directory in AWS.

When you submit a job from Pleiades, the value of the job's $PBS_O_WORKDIR variable (the **`cwd`** at the time of the **`qsub`** operation) is recreated in AWS. If the job uses your persistent /nobackup directory, it will be under that filesystem. Otherwise, the value will be a symbolic link created under the node's root filesystem that points to the primary job-time filesystem. This enables the job to change (**`cd`**) to the $PBS_O_WORKDIR directory.

For jobs submitted from Pleiades, the $PBS_O_WORKDIR is restricted to one under your /nobackup directory, since you can only submit batch jobs to AWS from a Pleiades /nobackup directory.

```
#CLOUD -stagein_file=path_to_file/file
#CLOUD -stagein_dir=path_to_dir
#CLOUD -stageout_file=path_to_file/file
#CLOUD -stageout_dir=path_to_dir
#CLOUD -stageout_file_delete=path_to_file/file
#CLOUD -stageout_dir_delete=path_to_dir
```

Note: In a PBS script, all PBS directives should be specified before the CLOUD directives.

More details about staging are described below:

Using the stagein Directive

You can **`stagein`** files or directories under your Pleiades /nobackup directory, but not under your Pleiades /home directory.

All Pleiades files to be used in AWS must be specified in the PBS script. You can give individual files (wildcards included) or individual directories.

TIP: If you have a lot of files to stage in, you can tar up the directory, and just stage in the tar file. This will speed up the process. (You will need to untar the file inside your PBS script.)
You can use either an absolute or a relative path. When a relative path is used, it is relative to the $PBS_O_WORKDIR directory on Pleiades where the **`qsub`** command was issued.

The staged-in files and/or directories will have the same directory structure on AWS as they have on Pleiades. In the two examples below, the PBS job is submitted from **`/nobackup/`*`username`*`/run`** on Pleiades.

Example 1

The following directive will result in **`/nobackup/username/run/input/file`** on Pleiades to be staged into **`/nobackup/username/run/input/file`** on AWS, *not* **`/nobackup/username/run/file`** on AWS:

```
#CLOUD -stagein_file=input/file
```

Example 2

The following directive will copy **`/nobackup/username/src/mpi_pi.f`** from Pleiades to **`/nobackup/username/src/mpi_pi.f`** on AWS:

```
#CLOUD -stagein_file=/nobackup/username/src/mpi_pi.f
```

Using the stageout Directive

Only the relative path should be used. In addition, staging out files to Pleiades at a relative path above $PBS_O_WORKDIR (such as ../src) is not supported. The staged out files and/or directories will have the same directory structure on Pleiades as they have on AWS.

Stageout and Delete

Optionally, you can use the following directives to perform two tasksâ   stage out, then delete:

```
#CLOUD -stageout_file_delete=output
#CLOUD -stageout_dir_delete=test_dir
```

Order of Operation

- **`-stagein_file`** and **`-stagein_dir`** operations occur in the PBS prologue.
- **`-stageout_file`** and **`-stageout_dir`** operations occur at the end of job, specifically:
    - For AWS filesystems with a distinct filesystem server, such as a persistent filesystem or a shared-type job-time filesystem, the operations occur on the filesystem server after the job exits.
    - For other types of job-time filesystems, the operations occur in the PBS epilogue.

If you are using other **`CLOUD`** directives, see Order of Operation of Cloud Directives.

Transferring Files Between a Pleiades or AWS Front End and AWS S3 Storage

This article describes how to use the **`nas_s3_xxx`** set of commands on a Pleiades front end (PFE) or an AWS dynamic front end to view, transfer, and delete files and directories under your AWS S3 root location (i.e., /).

Similar to the Lou mass storage system at NAS, the AWS S3 system functions as long-term storage. The data stored in your S3 folder is not accessible or visible to any other user. You will not have the required AWS credentials to directly access the S3 storage from either a PFE or an AWS dynamic front end, but you can use the following set of commands to check, transfer or delete files in your S3 folder from a PFE or from an AWS dynamic front end.

On Pleiades, these commands are located in the **`/u/scicon/tools/bin`** directory. On an AWS dynamic front end, they are located in the **`/opt/pbs/bin`** directory.

Note: To use these commands, you must have write permission on the location where the commands are issued.

- **View:** **`nas_s3_ls [--group`** ***`X`*** **`]`** ***`my_folder`***
- **Transfer to S3:** **`nas_s3_put [--group`** ***`X`*** **`] [`** ***`file1 directory1 file2 directory2`*** **`...]`** ***`s3_folder`***
- **Transfer from S3:** **`nas_s3_get [--group`** ***`X`*** **`] [`** ***`file1 folder1 file2 folder2`*** **`...]`**
- **Delete:** **`nas_s3_del [--group`** ***`X`*** **`] [`** ***`file1 file2 .... folder1 folder2`*** **`...]`**

If you have more than one AWS group, and you want to use a non-default one, add **`--group`** ***`X`*** where *X* is a group ID (GID).

Please note the following:

- A **`nas_s3_xxx`** command creates a PBS job to the **`s3op`** queue behind the scenes. It may take a few minutes to get a response back after issuing the command. After putting files/directories to S3 with the **`nas_s3_put`** command, it may take up to 10 minutes to see them at S3 with the **`nas_s3_ls`** command.
- Currently, there is a limit of 10 **`nas_s3_xxx`** jobs per user. Pending usage loads and network bandwidth, the limit may change in the future.

To learn more and find examples, simply issue any one of these commands without any options on a PFE or an AWS dynamic front end.

Transferring Files Between AWS S3 Storage and Your AWS PBS_O_WORKDIR Directory

Use the **CLOUD** directives described in this article to copy files between your S3 storage on AWS and your $PBS_O_WORKDIR directory in AWS.

When you submit a job from Pleiades, the value of the job's $PBS_O_WORKDIR variable (the **cwd** at the time of the **qsub** operation) is recreated in AWS. If the job uses your persistent /nobackup directory, it will be under that filesystem. Otherwise, the value will be a symbolic link created under the node's root filesystem that points to the primary job-time filesystem. This enables the job to change (**cd**) to the $PBS_O_WORKDIR directory.

Note: In a PBS script, all PBS directives should be specified before the **CLOUD** directives.

Get Files or Directories from S3 to $PBS_O_WORKDIR

The following **CLOUD** directives allow you to get files or directories from the specified S3 folder to the $PBS_O_WORKDIR directory in AWS.

- To get your ***dir1*/*dir2*/file** in S3 and place it as **PBS_O_WORKDIR/*dir1*/*dir2*/file**:

  ```
  #CLOUD -get_file=dir1/dir2/file
  ```
- To get your ***dir1*/*dir2*/file** in S3 and place it as **PBS_O_WORKDIR/*dir2*/file**:

  ```
  #CLOUD -get_file=dir2/file:dir1
  ```
- To get your **dir1/dir2/file** in S3 and place it as **PBS_O_WORKDIR/file**:

  ```
  #CLOUD -get_file=file:dir1/dir2
  ```
- To get all files from your **dir1/dir2** directory in S3 and place them under **PBS_O_WORKDIR/*dir1*/*dir2***:

  ```
  #CLOUD -get_dir=dir1/dir2
  ```
- To get all files from your **dir1/dir2** directory in S3 and place them under **PBS_O_WORKDIR/*dir2***:

  ```
  #CLOUD -get_dir=dir2:dir1
  ```
- To get all files from your **dir1/dir2** directory in S3 and place them under **PBS_O_WORKDIR**:

  ```
  #CLOUD -get_dir=/:dir1/dir2
  ```

Example 1

The following command line will get the output.dat file under your S3 **run01** directory and place it as **$PBS_O_WORKDIR/run01/output.dat**:

```
#CLOUD -get_file=run01/output.dat
```

Example 2

The following command line will get all files under your S3 **run02/data** directory and place them under **$PBS_O_WORKDIR/data**:

```
#CLOUD -get_dir=data:run02
```

Put Files or Directories to S3

To put files or directories to S3, run the following commands, where **`src_file`** or **`src_dir`** is relative to **`$PBS_O_WORKDIR`** and **`optional_dest_dir`** is relative to your S3 root location.

```
#CLOUD -put_file=src_file:optional_dest_dir
#CLOUD -put_dir=src_dir:optional_dest_dir
```

If **`optional_dest_dir`** is not specified, **`#PBS_O_WORKDIR/src_file`** is copied to your S3 root location, and **`$PBS_O_WORKDIR/src_dir/*`** is copied to the **`src_dir/*`** under your S3 root location.

Example 3

To put **`$PBS_O_WORKDIR/data/output.dat`** as **`run02/output.dat`** under your S3 root location:

```
#CLOUD -put_file=data/output.dat:run02
```

Example 4

To put **`$PBS_O_WORKDIR/data/hdf5/*`** as **`run03/hdf5/*`** under your S3 root location:

```
#CLOUD -put_dir=data/hdf5:run03/hdf5
```

Order of Operation

The **`-get_xxx`** operations occur before the execution of the body of the PBS script, while the **`-put_xxx`** operations occur after the execution of the body of the PBS script.

If you are using other **`CLOUD`** directives, see Order of Operation of Cloud Directives.

Creating AWS Job-Time Filesystems

A job-time filesystem is created at the beginning of each PBS batch job submitted from either a Pleiades front-end system (PFE) or an AWS dynamic front end. The filesystem is maintained while the job is running. When it is terminated at the end of the job, the data in the filesystem is lost. You are charged for the size provisioned for the lifetime of the job and the uptime of the filesystem server (if needed).

Note: Job-time filesystems are not created for interactive PBS jobs submitted from an AWS dynamic front end.

Adding the CLOUD Directives to Your PBS Script

Use the following directives in your PBS script to create various types of job-time filesystems, and to specify the size and mount point:

```
#CLOUD -volume_type=type
#CLOUD -volume_size=size_in_GB
#CLOUD -volume_mount=/path/to/mount/on
```

Important: The CLOUD directives must be placed *after* the the PBS directives in a PBS script.

-volume_type

The **-volume_type** directive must be placed before the **â volume_size** and **â volume_mount** directives in the PBS script; before **â volume_put** and **â volume_get**, which are described in this article; and before **â volume_primary**, which is described in this article.

See the next section for a list of options you can specify for the **â volume_type** directive.

-volume_size

For **-volume_size**, a number in gigabytes (G) is expected; any unit you add to it is ignored. For example, each of the following three specifications results in a request for a volume size of 10 G:

```
#CLOUD -volume_size=10
#CLOUD -volume_size=10G
#CLOUD -volume_size=10T
```

-volume_mount

For **-volume_mount**, mounting as **/home** is not currently allowed.

You can use as many sets of job-time filesystems as you want for each job. However, to reduce complexity for your job, we recommend using a maximum of two.

Types of Job-Time Filesystems

There are several types of job-time filesystems you can configure for your job. Some factors to consider when choosing a type are I/O pattern, size, performance, and cost.

You can specify the following options for the **â volume_type** directive:

- **ephemeral**

  Uses the ephemeral Non-Volatile Memory Express (NVMe)-based Solid State Drive (SSD) space that is local to various computing instance types. This type will ignore the **â volume_size** setting because the local space determines the size available.

  Not all instance types have a local disk. If the type you specify doesn't have disks available, it ignores the directive.

  For best results using NVMe-based SSD, add **arch=c5d** into the **select** line in the PBS script, as that is the only instance type currently supported by NAS. Note that **c5d** comes in six sizes, with 1, 2, 4, 8, 19, and 36 physical cores, and up to 1,800 GB of NVMeâ based SSD.
- **local**

  Each computing instance gets its own unique EBS filesystem. The size of the EBS volume unique to each computing instance is determined by **â volume_size=*x***. The *x* GB EBS volume accessed by one computing instance cannot be accessed by another computing instance, which has its own *x* GB of EBS to use.
- **headnode**

  Only the head node gets the EBS volumes. The size of the EBS is determined by the **â volume_size**. Other computing instances do not have access to the EBS on the head node.
- **node=*x***

  The *x*th computing instance gets an EBS filesystem. Specifying **node=0** has the same result as specifying **headnode**. The EBS filesystem of this *x*th instance is not accessible by another instance.
- **shared**

  Creates a distinct filesystem server instance that NFS-exports the volumes to all computing instances. None of the computing instances is used as the filesystem server.

  To select an instance as a filesystem server, it first looks at the **â volume_size** request. If there is an instance type that has at least the amount of local spaceâ NVMe SSD or Hard Disk Drive (HDD)â requested, that instance will be selected over other instances with EBS mount storage. Otherwise, a c5.18x instance with added EBS volumes to satisfy the required space will be selected.

  In other words, this "shared" filesystem has a few options for instances it will select behind the scenes. For the following requested volume sizes:

    - Under 1.8 TB: a c5d.18x instance with local SSD disks is selected.
    - Between 4 TB and 16 TB: an h1.16x instance with local HDD disk is selected.
    - For any other size: the normal c5.18x instance with EBS volumes is selected.
- **headnode**

  The EBS volumes on the head computing instance are NFS-exported to all other computing instances.

  Note: Exporting ephemeral space, if any, on the head computing instance to other computing instances is currently not supported by the NAS cloud developers.
- **lustre**

Creates a distinct filesystem using Amazon FSx for Lustre. The size will be determined by the value of **`-volume_size`**. There is an AWS-imposed minimum size of 3,600 GB. This filesystem will be mounted on all compute nodes at the mount point specified by **`â  volume_mount`**.

Note: Using this filesystem will delay job startup by a few minutes while the filesystem is created.

For examples of how these directives are used, see this article.

Transferring Files Between AWS S3 and Job-Time Filesystems

Use these **`CLOUD`** directives to copy files between your job-time filesystems on AWS and your S3 storage on AWS.

```
#CLOUD -volume_put=s3_folder_name_to_copy_to
#CLOUD -volume_get=s3_folder_name_to_copy_from
```

Please note the following:

- All **`CLOUD`** directives should be placed after the PBS directives in a PBS script.
- The **`-volume_put`** and **`-volume_get`** directives should be placed after the **`â  volume_type`** directive in a PBS script, and can be placed either before or after the **`â  volume_size`** and **`â  volume_mount`** directives of the corresponding job-time filesystem.
- No leading slash should be used in the **`s3_folder_name`**, as demonstrated in the example in this section.
- The S3 folders are all relative to your S3 root location (i.e., /). Other users, even those in the same group, cannot see your files in S3.
- The **`-volume_get`** directives will get everything inside the specified folder. Getting a single file from an S3 folder is not currently supported.

If you give the **`-volume_put`** directive for a job-time filesystem where multiple nodes have their own space (such as the *ephemeral* or *local* type of filesystem, but not the *shared* type), then files from all of the nodes will be put in the same S3 folder. This results in files with the same name on each node ending up in an undetermined state as to which node's version is stored. You can avoid this by including **`{node}`** in the S3 folder name, so the path will have **`nodeX`** appended to it when saved on the *X* node's copy. For example, for a job that asks for three nodes with these directives:

```
#CLOUD -volume_type=local
#CLOUD -volume_put=run12/{node}
#CLOUD -volume_mount=/data
```

the following directories in your S3 environment will be created and files in the **`/data/`** directory from each node will be stored in the appropriate one:

```
/run12/node0
/run12/node1
/run12/node2
```

The **`-volume_get`** directive works similarly in these situations, but without the need for the **`{node}`** text.

For a filesystem type that is unique to each node (for example, ephemeral, local, headnode, or node=*X*), if the number of nodes used in the 'put' operation is less than the number of nodes used in the 'get' operation, then some nodes may have empty directories from the 'get' operation. However, if the number of nodes used in the 'put' operation is more than those in the 'get' operation, then not all the data from the 'run12' S3 bucket will be brought back.

For more information, see:

- Examples of Job-Time Filesystem Related Directives
- Optional Job-Time Filesystem Related Directives

Order of Operation

- The `-volume_get` operation happens when the compute instance boots.
- The `-volume_put` operation for shared filesystems occurs after the job exits, and therefore will not show up in the output of `â   volume_list` (if included in the job script). You can use the `nas_s3_ls` command (described in this article) to see them in S3 afterwards.
- The `-volume_put` operation for non-shared filesystems occurs after the PBS script but before the `â   volume_list` operation.

If you are using other `CLOUD` directives, see Order of Operation of Cloud Directives.

Optional Job-Time Filesystem Related Directives

This article describes optional AWS Cloud directives to use in addition to those for creating a job-time filesystem and those used for transferring files between S3 and a job-time filesystem.

- **`-volume_include`**

  By default, jobs will not have persistent, shared /home and /nobackup filesystems mounted if there are any **`â volume_type`** directives given. Without a persistent /home filesystem mounted, you will get an empty /u/username directory as part of the root filesystem on the node.

  The **`-volume_include`** option can be used to indicate that you want to have the persistent /home and/or /nobackup mounted as normal.

  ```
  #CLOUD -volume_include=home
  #CLOUD -volume_include=nobackup
  ```

  Important: If the persistent /home or /nobackup filesystems are mounted, those two paths are not available as a mount destination for job-time filesystems.

  The **`-volume_include`** directive, together with **`â stagein_file/dir`** and **`â stageout_file/dir`** used to stage in/out files between Pleiades and AWS can be placed above the first **`â volume_type`** directive.
- **`-volume_home_{bashrc,profile,login,cshrc}`**

  ```
  #CLOUD -volume_home_bashrc=local_file
  #CLOUD -volume_home_cshrc=local_file
  #CLOUD -volume_home_profile=local_file
  #CLOUD -volume_home_login=local_file
  ```

  These directives will take the given file in the current directory (i.e., $PBS_O_WORKDIR) and make it the ~/.bashrc (or .cshrc, .profile, .login, depending on which specific directive is used) to be automatically sourced under $HOME at job startup for all instances. In the event there isn't an existing persistent shared /home used, the .bashrc will be placed under each instance's empty /home under the root filesystem. This will allow some control over the environment for things that can't be set in the PBS script (i.e., variables on the nonâ headnode).

  These directives can be placed above the first **`â volume_type`** directive.

  Note: The default shell is bash unless you have requested changing it. If you want to use csh and have the .cshrc sourced in a PBS job, add the following to the beginning of your PBS script:

  ```
  #PBS -S /bin/csh
  ```
- **`-volume_primary`**

  In the event that there are multiple filesystems mounted in a job, you can use this directive to toggle a filesystem on as the source and destination for **`â get_file`**, **`â put_file`**, **`â stagein_file/dir`**, and **`â stageout_file/dir`**, and also to create a softlink for $PBS_O_WORKDIR to the following primary filesystem:

  ```
  #CLOUD -volume_primary
  ```

  This directive should be placed anywhere before the specification of the next filesystem block that starts with **`â volume_type`**.

However, if a persistent /nobackup filesystem is mounted, it is set as the primary regardless of whether any job-time filesystem has **`-volume_primary`** set. Conversely, a persistent /home filesystem cannot be set as primary.

If more than one job-time filesystem is specified as the primary, the first one specified with **`â   volume_primary`** is chosen.

If there is no user-indicated primary, the system picks the first one that exists in this order:

```
shared
lustre
headnode_shared
headnode
node=X
local
ephemeral
```

For example, the following specification will result in */shared_1* as the chosen primary.

```
#CLOUD -volume_type=headnode
#CLOUD -volume_mount=/data
..
#CLOUD -volume_type=shared
#CLOUD -volume_mount=/shared_1
..
#CLOUD -volume_type=shared
#CLOUD -volume_mount=/shared_2
```

- **`-volume_list`** and **`-volume_delete`**

These two options can be used to see the contents and to remove, if desired, any previously 'put' filesystems. They can be placed anywhere after the #PBS directives.

```
#CLOUD -volume_list=s3_folder_to_list
#CLOUD -volume_delete=s3_folder_to_remove
```

The **`-volume_list`** and **`-volume_delete`** operations occur after the execution of the PBS script. Read this article for more information.

Examples of Job-Time Filesystem Related Directives

This article provides a few examples of how the job-time related CLOUD directives are used.

Please review the following articles before you begin using the examples shown below:

- Creating a job-time filesystem
- Transferring files between S3 and a job-time filesystem
- Optional job-time filesystem related directives

Using Local Job-Time Filesystems

Data written to a local disk of a node are not accessible to processes running on other nodes. One scenario where the usage of such local disks is appropriate: running a CFD application where its MPI processes write out one file per rank or per node and the file is not needed by other ranks or nodes. The following examples walk you through this scenario.

Examples 1 and 2 show how to utilize the job-time local disks (/data, shown in the examples) for staging in the executable file (a.out) and input file (input.dat) from the Pleiades /nobackup system; and how to store rank-identified restart files generated by individual ranks in separate node-identified sub-folders of AWS S3 at the end of a job.

In Example 1, the ephemeral volume type is used when the local space needed for the restart file is less than 1.8 TB. In Example 2, the local volume type is used when the space needed is more than 1.8 TB.

In Example 3 (after you have followed the steps in Example 1 or Example 2), the restart files are loaded back from the S3 sub-folders to the local disk to resume simulation.

Note: If you do not store the restart files into sub-folders according to node numbers, when it is time to load the files back for the next job, each node will get all the restart files instead of just the ones needed for the ranks on the node.

Example 1

Specifying arch=c5d allows c5d.18x instances with 1.8 TB of NVMe SSD to be used as local disks.

```
#PBS -l select=10:mpiprocs=36:arch=c5d
...
#CLOUD -volume_type=ephemeral
#CLOUD -volume_put=outputfolder/{node}
#CLOUD -volume_mount=/data
..
#CLOUD -stagein_file=a.out
#CLOUD -stagein_file=input.dat
..

cd /data/

mpiexec -np 360 ./a.out < input.dat
```

Example 2

If you space needs more than 1.8 TB of space for the restart files, then specify the local volume type and a large volume size (for example, 2,048 GB).

```
#PBS -l select=10:mpiprocs=36:arch=c5d
...
#CLOUD -volume_type=local
#CLOUD -volume_put=outputfolder/{node}
#CLOUD -volume_mount=/data
#CLOUD -volume_size=2048
..
#CLOUD -stagein_file=a.out
#CLOUD -stagein_file=input.dat
..

cd /data/

mpiexec -np 360 ./a.out < input.dat
```

Example 3

Since the **`-volume_put`** directive was included when the previous job was done (as shown in examples 1 and 2), the files got saved off for this follow-on job to start from.

The **`-volume_get`** directive will bring back the files into the corresponding node in this new job. You do not need to include **`{node}`** in the **`get`** directive.

```
#PBS -l select=10:mpiprocs=36:arch=c5d
...
#CLOUD -volume_type=ephemeral
#CLOUD -volume_get=outputfolder
#CLOUD -volume_put=outputfolder
#CLOUD -volume_mount=/data
..
#CLOUD -stagein_file=a.out
#CLOUD -stagein_file=input.dat
..
cd /data/

mpiexec -np 360 ./a.out < input.dat
```

Using Shared Job-Time Filesystems

Example 4 shows how to create a filesystem (/nobackup, in this example) that only exists for the duration of the job and is shared by all nodes.

Example 4

```
#PBS -l select=10:mpiprocs=36
...
#CLOUD -volume_type=shared
#CLOUD -volume_size=2048
#CLOUD -volume_mount=/nobackup
..

cd /nobackup

mpiexec -np 360 ./a.out < input.dat
```

The directives **`-volume_put`** and **`-volume_get`** can be used if you want to save files in S3 between runs.

Example 5

If your job is such that the head node/rank does most of the writing, but there are some files that all ranks need to see, the **`headnode_shared`** volume type is useful.

```
#PBS -l select=10:mpiprocs=36
..
#CLOUD -volume_type=headnode_shared
#CLOUD -volume_size=2048
#CLOUD -volume_mount=/nobackup
...
```

Using Both Shared and Local Job-Time Filesystems

Example 6

Certain jobs may have some ranks that perform CFD computations and some that handle I/O or mesh regridding and need local disk space. You can use a shared filesystem for the computation and the node=X volume for those ranks that need local disk space. For example, if rank 10 and rank 50 need space but not the others, you can use:

```
#PBS -l select=10:mpiprocs=10
...
#CLOUD -volume_type=shared
#CLOUD -volume_size=100
#CLOUD -volume_mount=/nobackup
...
#CLOUD -volume_type=node=1
#CLOUD -volume_size=512
#CLOUD -volume_mount=/data
..
#CLOUD -volume_type=node=5
#CLOUD -volume_size=512
#CLOUD -volume_mount=/data
```

In this case, only node 1 and node 5 will have extra space in /data. The other nodes will use the shared filesystem, /nobackup.

Using a Non-Default Startup Script

Example 7

A job startup script under $HOME is useful to provide a customized environment for a login session or for different nodes in a PBS job. The customized environment may include the module load commands or set certain environment variables.

If the job does not use the persistent /home shared filesystem, where a startup script may be available, you can provide one that is accessible from the job's $PBS_O_WORKDIR, to be used in the place of the $HOME startup script. You can tell PBS to use that file instead of the system default ~/.bashrc file created on each compute node at boot time, as follows:

```
...
#CLOUD -volume_home_bashrc=my_bashrc_file
...
```

Order of Operation of Cloud Directives

In situations where you have files with the same filename but with different contents in different places--such as on Pleiades, various folders of AWS S3, and various job-time filesystems--pay attention to the executation order shown below to avoid unintentionally getting or putting, or using or deleting the wrong copy.

Please review the following articles to understand the overall operation of the HECC cloud environment:

- Staging between Pleiades and AWS within PBS
- Transferring files between AWS S3 and $PBS_O_WORKDIR
- Creating a job-time filesystem
- Transferring files between AWS S3 and job-time filesystems
- Optional job-time filesystem related directives

Order of Execution

- Persistent filesystems:
    1. -stagein_file/_dir
    2. -get_file
    3. -volume_home_{bashrc,profile,login,cshrc}
    4. pbs script
    5. (-put_file/-stageout_file/_dir)
    6. -volume_list
    7. -volume_delete
- "Shared" type job-time filesystems:
    1. -volume_get
    2. -stagein_file/_dir
    3. -get_file
    4. -volume_home_{bashrc,profile,login,cshrc}
    5. if primary then -stagein_file/_dir and -get_file
    6. pbs script
    7. -volume_list
    8. -volume_delete
    9. -put_file
    10. -stageout_file/_dir
    11. -volume_put
- Non-"shared" type job-time filesystems (node, local, ephemeral, master_shared):
    1. -volume_get
    2. -volume_home_{bashrc,profile,login,cshrc}
    3. if primary then -stagein_file/_dir and -get_file
    4. pbs script
    5. -volume_put
    6. (-put_file/-stageout_file/_dir)
    7. -volume_list
    8. -volume_delete

The (-put_file/-stageout_file/_dir) is in the order that the directives exist in the PBS script.

The -volume_get operation happens when the compute instance boots up. For the -volume_put operation, it depends on the filesystem used:

- "Shared" filesystems: Done on the filesystem server after the job exits.

- All other filesystems: Done on the filesystem after the PBS script and before the -volume_list operation.

WARNING: It is recommended not to delete a folder in which you intend to 'put' files within the same PBS job.

Cloud Billing Units and Accounting

Usage on the HECC AWS Cloud is charged in Cloud Billing Unit (CBU) where 1 CBU = 1 US dollar. A project is given an allocation of *N* CBUs when the principal investigator (PI) provides his/her NASA Work Breakdown Structure (WBS) number with *N* US dollars.

Charges include the following categories:

- ### AWS Dynamic Front End Usage

  A charge to the user for using the AWS dynamic front end is recorded on a per-front-end-request basis. After the front end session is terminated, an email with the charging record is sent to the user (see AWS dynamic front end for an example email). This charge is reflected in the **`acct_query`** output under the frontend queue.
- ### PBS Jobs Usage

  A charge to the user for running a PBS job is recorded on a per-job basis. After the job is terminated, the charging record is included in the PBS output file, which is sent to the $PBS_O_WORKDIR. For jobs submitted from a PFE, the $PBS_O_WORKDIR is under the user's Pleiades /nobackup directory. For jobs submitted from an AWS dynamic front end, the $PBS_O_WORKDIR is a directory on an AWS filesystem (a sample output can be found at the end of this article). This charge is reflected in the **`acct_query`** output under the cloud_exec queue.
- ### nas_s3_{get,ls} and Per-User S3 Long Term Storage Usage

  Charges associated with per-user and/or per-job S3 usage such as running the **`nas_s3_{get,ls}`** commands from NAS (which incur both file-transfer-out of AWS and network costs) and storing data in S3 are recorded daily with no individual notification in the form of an email or per-job output. They are reflected in the **`acct_query`** output under the daily resource charge (drc) queue.
- ### PBS Server at AWS, Job-Time Filesystem of -volume_type=shared, Persistent Filesystems, and Project-Level S3 Storage Usage

  A charge to the PI for using resources that cannot be accounted for on a per-job and/or per-user basis is recorded on a per-day basis. The record of these charges is only available through **`acct_query`** under the drc queue for the PI.
- ### AWS Project Account Charge

  In order to comply with NASA rules and regulations for working in the cloud, there are a number of underlying resources active in the project's AWS account. These cannot easily be included on a per-PBS-job or per-day basis, so they are added at the end of the month when the final AWS bill arrives. The charge normally amounts to ~$100/month and it is not reflected in the **`acct_query`** output.
- ### NAS Overhead

  In addition to the hardware and network resources, there is support infrastructure by AWS and HECC to be maintained. An overhead cost for this infrastructure is added to the charges for the above items.

Allocation Thresholds

Allocations are checked once a day at 12:05 a.m. (Pacific Time). When a project's allocation goes below a certain threshold, the GID and its users are disabled in the PBS Access Control List (ACL). Re-enabling happens automatically at 12:05 a.m. if the allocation goes above the threshold after the PI sends in additional funding. In the event the PI wants his/her account to be reactivated immediately, NAS staff with root privilege can also run a script at anytime to update the ACL.

Use the **`acct_ytd`** and **`acct_query`** tools on a PFE to check your allocation usage on AWS (**`-caws`**) or all supported HECC Cloud providers (**`-ccloud-all`**). For example,

```
pfe% acct_ytd -caws
pfe% acct_ytd -ccloud-all
pfe% acct_query -caws -u username -p your_gid -b 07/01/19
pfe% acct_query -ccloud-all -u all -p all -b 07/01/19
```

A sample output of the last command is shown here:

```
Printing information for all the users supplied.
REPORT FROM 07/01/19 TO 07/19/19

ACCT_QUERY: Generating data ...


GRAND TOTAL FOR 07/01/19 TO 07/19/19
CLIENT    USER      PROJECT    QUEUE     SBU Hrs/Cloud BU
------- ---------- --------- ----------- ----------------
aws      staff1     cstaff    cloud_exec    2.207
aws      staff1     cstaff    drc           6.000
aws      staff1     cstaff    frontend     73.781
aws      staff2     cstaff    cloud_exec    2.337
aws      staff2     cstaff    drc           0.000
aws      staff2     cstaff    frontend      5.821
TOTAL FOR    aws.cstaff                    90.146

TOTAL FOR ALL PROJECTS FOR CLIENT: aws     90.146

TOTAL FOR PROJECT ON ALL CLOUD: cstaff     90.146
```

Note: All numbers shown in the output of **`acct_ytd`** and **`acct_query`** for HECC AWS Cloud usage are in CBUs.

AWS PBS Resources and Examples

This article lists the **qsub** command options that are specific to the HECC AWS Cloud, and provides examples showing how to use them.

Accepted Resource Attributes

You can use these attributes for the **-l select** option:

```
ncpus=n   : minimum n cores per node

mpiprocs=n: n MPI processes per node

mem=nGB   : minimum n GB of RAM per node; can be simply nG, ng, or nM, nm.

arch=xx   : specifies an architecture type

             If not specified, the cheapest instance type
             that satisfies the user's request will be
             chosen automatically.

             most common ones are: c4/c5/m4/m5
             c4 allows for up to 18 cores and  60G per node
             c5 allows for up to 48 cores and 192G per node
             m4 allows for up to 32 cores and 256G per node
             m5 allows for up to 48 cores and 384G per node

gpu=xxx   : To specify the GPU node type.
             xxx should be either k80 or v100.
             The K80 nodes have either 1, 8, or 16 GPUs per node
             The V100 nodes have either 1, 4 or 8 GPUs per node

ngpus=n   : minimum n gpus per node

net=x     : Target network bandwidth in/out of node.
             Options supported are 1, 10, 25.
             Not specified will allocate a node type based
             on other specified resource requests only.
             If specified will allocate (if possible)
             a node type that satisfies other requests and:
              net=1  -> can sustain around 1+ Gbps,
              net=10 -> can sustain 10 Gbps
              net=25 -> can sustain 25 Gbps
```

The following PDFs list the CPU and GPU node types that are supported in AWS:

- CPU node types (PDF)
- GPU node types (PDF)

In all regions of AWS, on-demand nodes are used by default. Spot nodes are generally significantly less expensive then on-demand nodes. The drawback is that spot nodes are subject to being taken away based on AWS load and are not always available.

To request spot nodes for your cloud job, use the **cloud_model** directive in your PBS script:

```
#CLOUD -cloud_model=[spot, onlyspot]:block
```

- Specifying the type "**spot**" will indicate that the job should start using spot instances; if there are not enough spot resources for the job, it will transfer over to an on-demand job type.
- Specifying the type "**onlyspot**" will indicate that the job should start using spot instances; if there are not enough spot resources, the job will fail.
- The optional value "**:block**" indicates you want all instances to fail if any single instance is taken away.

For example, the following line will try to start the job on spot instances; if that isn't possible, it will try to start them as on-demand instances. If any of the spot instances is taken away by AWS, the entire job will be terminated:

```
#CLOUD -cloud_model=spot:block
```

On-demand jobs can also fail if resources are not available in AWS. Jobs are automatically retried 10 times before failing due to insufficient resources.

Note: In the PBS script, a **`CLOUD`** directive should be placed after all PBS directives have been specified.

Job Dependencies

We do not recommend submitting a job (*Job B*) that depends on another job (*Job A*), such as **`-W depend=afterany:`*`job_id.server_name`*`.nas.nasa.gov`**, because resources for *Job B* may be grantedâ and charging for *Job B* may beginâ before *Job A* terminates or completes, and before *Job B* actually executes.

Rerunning Jobs

Jobs submitted to run on the AWS cloud cannot be rerun by PBS, no matter what causes the job to fail. Specifying **`#PBS -r`** has no effect.

Resource Request Examples

- To request 2 nodes with a minimum of 2 cores and 16g of memory:

  ```
  #PBS -l select=2:mpiprocs=2:ncpus=2:mem=16g
  ```
- To request 3 c4 nodes each with a minimum of 18 cores and 48g of memory:

  ```
  #PBS -l select=3:ncpus=18:mem=48G:arch=c4
  ```
- To request 2 nodes with at least 1 K80 cards per node:

  ```
  #PBS -l select=2:mpiprocs=2:ncpus=2:mem=16g:gpu=k80:ngpus=1
  ```
- To request 3 nodes each with a minimum of 14 cores and 4g of memory, plus 2 nodes each with a minimum of 13 cores and 0.5G of memory, and 1 node with at least 3 K80 cards:

  ```
  #PBS -l select=3:mem=4g:ncpus=14+2:mpiprocs=13:mem=512M+1:gpu=k80:mem=16G:ngpus=3
  ```
- To request 8 nodes, each with 8 V100 GPUs:

  ```
  #PBS -l select=8:mpiprocs=2:ncpus=2:mem=16g:gpu=v100:ngpus=8
  ```

Using Jupyter Notebook or JupyterLab in the AWS Cloud

You can run a Jupyter notebook in an AWS front-end instance or in a PBS batch job, with either GPU or non-GPU instances.

Before You Begin: To use this capability, you must set up SSH passthrough and SSH port forwarding from your local system to a Pleiades front-end system (PFE) and then to an AWS node. If you need help to set these up, send an email with your contact information to support@nas.nasa.gov.

Running a Jupyter Notebook or JupyterLab in an AWS Front-End Instance

When you start an AWS front-end instance, add either the **`--jupyter`** or the **`--jupyterlab`** argument to the **`aws_fe`** command. For example, to start a Jupyter notebook or JupyterLab on an instance with a single NVIDIA Tesla V100 GPU card, run:

```
pfe: module load scicon/aws_cli_tools
pfe: aws_fe --group cloud_gid --hour 5 --gpu=v100 --jupyterlab

or

pfe: aws_fe --group cloud_gid --hour 5 --gpu=v100 --jupyterlab
```

When the instance starts, you will be sent an email with instructions describing how to connect and point your local browser to the Jupyter login page.

By default, a token is used for authentication when connecting to the Jupyter web interface. You can obtain this token by running the following command on the AWS front-end instance:

For Jupyter Notebooks:

```
aws: jupyter notebook list
```

For JupyterLab:

```
aws: jupyter server list
```

If you prefer to set up a password instead, you can create one while logged into any AWS instance by running the following command:

```
aws: jupyter notebook password
```

You will be prompted for a password, and the file **`jupyter_notebook_config.json`** will be created in the **`/u/username/.jupyter`** directory. (If this file already exists, an entry will be added to it.)

The most commonly used packages are already installed in the version of Python that Jupyter uses by default. If you want to use different packages, you can set up your own virtual environment for a kernel. For example, because GPU-capable versions of the Python packages are not installed in the default environment, you might want to create your own.

Creating Your Own Virtual Environment

The example below shows how to create a virtual environment to install GPU-capable versions of the Python packages.

TIP: To see a list of Python packages you might want to install, run the **`module help cuda/10.0`** command.
To create your own virtual environment, run:

```
aws: module purge ; module load python/Python3.6
aws: python3 -m venv /u/username/.venv/gpu
aws: module purge
aws: module load cuda/10.0
aws: source /u/username/.venv/gpu/bin/activate
aws(gpu): pip install .. whatever packages you need...
aws(gpu): pip install ipykernel
```

Then, enable Jupyter to see this virtual environment as an option for its kernel by creating this file:

```
/u/username/.local/share/jupyter/kernels/name/kernel.json
```

where *name* can be anything, such as the name of your environment. The file should contain the following contents:

```
    {
     "argv": [
      "/u/username/.venv/gpu/bin/python",
      "-m",
      "ipykernel_launcher",
      "-f",
      ""
     ],
     "display_name": "My GPU",
     "env": {"LD_LIBRARY_PATH":"/nasa/cuda/10.0/lib64:/nasa/cuda/10.0/extras/CUPTI/lib64:
                /nasa/cuda/10.0/targets/x86_64-linux/lib:/nasa/cuda/10.0/TensorRT/lib"},
     "language": "python"
    }
```

Notes:

- The **`"env"`** line in the .json file is shown on two lines due to a KB formatting issue. It should be on one line.
- You must load the **`python3.6`** module because the default Python version (v3.7) has issues with the TensorRT packages required for **`tensorflow-gpu`** on the system.
- The first item in the **`argv`** list is the full path to Python in your virtual environment (**`venv`**). Be sure to use **`/u/username/`** instead of `'~/'`.
- Set the value for **`display_name`** to anything you like.
- Set the LD_LIBRARY_PATH variable to point to the CUDA libraries if you installed **`tensorflow-gpu`**, which needs these libraries. CUDA v10 is the only version that works with the current version of **`tensorflow-gpu`** (as of April 2020).

When Jupyter is started, the Python virtual environment (**`venv`**) will be available as a kernel option under the **New** dropdown menu.

Running a Jupyter Notebook in a PBS Batch Job

You can start an instance with Jupyter running in a PBS batch job, either from a PFE or an AWS front end. Jupyter running in a batch job will have access to any Python virtual environments and password settings you've added.

In your **`qsub`** command line, add the PBS **`select`** option **`service=jupyter`** to the first node.

For example, from a PFE:

```
pfe: qsub -q cloud@clpbs-01 -l
```

```
    select=1:mpiprocs=14:mem=24:service=jupyter+2:mpiprocs=14:mem=24 script.pbs
```

Note: The **qsub** command line is shown on two lines due to a KB formatting issue. It should be on one line.

From an AWS front end:

```
aws: qsub -l select=1:mpiprocs=14:mem=24:service=jupyter+2:mpiprocs=14:mem=24 script.pbs
```

Note: Jupyter is only started on the head node, even in the following two submission examples:

- **`qsub -l select=3:mpiprocs=14:mem=24:service=jupyter`**
- **`qsub -l select=1:mpiprocs=14:mem=24:service=jupyter+2:mpiprocs=14:mem=24:service=jupyter`**

For the following use cases, Jupyter will look at PBS_NODEFILE and spawn onto the other nodes:

- If you are using the built-in Dask as the parallel processing tool.
- If you are using any Python packages or tools that understand the Slurm or PBS environments.

To get access to the Jupyter notebook running inside the batch job, be aware that batch job instances are not accessible from outside AWS, but they are accessible from a front end running in AWS. From an AWS front end, there are two possible ways to obtain the URL needed to access Jupyter running on the head node of your batch job.

- If one of the following two files is successfully deposited by the Jupyter server to your $HOME on AWS, look for the URL embedded in that file:
    - ~/jupyter_output_${PBS_JOBID}
    - ~/jupyter_lab_output_${PBS_JOBID}

  Note that this method works only if the compute nodes of your batch job sees your AWS $HOME as a default shared filesystem.
- Obtain the URL by running the **`jupyter notebook list`** command on the head node of your batch job by following these steps:

1. On an AWS front end, use the **qstat** command as shown below to list the PBS jobs running and the head node associated with the job:

   ```
   aws: qstat -W o=+Rank0
   ```
2. Use SSH to connect to the head node in order to access the Jupyter notebook running inside the batch job.
3. To get the URL needed to access Jupyter, run the following command on the batch node:

   ```
   aws: jupyter notebook list
   ```

Running a Singularity Container Image in AWS

In our Amazon Web Services (AWS) environment, Singularity is built into the base compute image (operating system), so you do not need to load a module in order to run a Singularity container in AWS. Normal operation is nearly identical to the Pleiades environment described in Running a Singularity Container Image on Pleiades, with one main difference described below.

Building/Converting SIF Files in AWS

If you want to build your own SIF files, or convert an SIF file into a sandbox image in the AWS environment, you must add the **`container=singularity`** option to the typical PBS **`â  l select`** command line, as follows:

```
#PBS â  l select=2:mpiprocs=16:container=singularity:mem=64
```

This option increases the size of the /tmp filesystem and points the Singularity cache and TMPDIR environment variables to this filesystem. This is helpful both when you build SIF images and when you run them.

Note: When you start a front end in AWS using the **`aws_fe`** command, the **`â  â  container singularity`** command line argument provides the same functionality as the **`container=singularity`** does in the **`â  l select`** line.

For more information, see AWS Dynamic Front End.

AWS Cloud FAQs

Can I get more information about your AWS Public Cloud offering?

To learn more about our AWS Public Cloud offering, please watch the 50-minute webinar recording "Overview of HECC Pay-for-Use AWS Cloud," which you can find on the HECC Webinars page.

Also, see our many Knowledge Base articles in the AWS Cloud section.

On a high level, what kinds of work are supported?

The work must be an HPC science or engineering project, whose workflow can be modified to run via a `qsub` command (in order to submit a batch or interactive job from a NAS front-end system).

Note: A batch job running on the AWS compute instances will have no access to/from NAS or to/from the internet. If your workflow requires the AWS instance be able to access the internet (but not be accessible from the internet), an AWS front-end instance can be launched. Access from NAS to this instance is allowed.

On a high level, what kinds of work are not supported?

We are not currently able to support the following scenarios:

- Enterprise applications that are not HPC-related (for example, Tecplot, MATLAB, and ANSYS are supported while web or eCommerce apps are not).
- Enterprise-level data storage (for example, offsite backups of data via appliances, unless it's a NAS-owned appliance).
- AWS services that don't yet have FedRamp certificationsâ and the demand for a given service is not large enough to justify a request to the NAS security group to allow it.
- User access to the AWS console or command-line interface. (Only certain NAS staff with special privileges can access these.)
- AWS native container technologies.
- Access to resources and/or job submissions from the internet at this time.
- Publicly accessible web pages/S3 storage. This may be allowed in the future.
- Direct access to resources from outside of NAS. This restriction may be removed in the future.

Please check back, as some of these scenarios may be supported in the future.

What CPU and GPU node types are supported in AWS?

The following PDF lists show the CPU and GPU node types that are supported in AWS:

- CPU node types (PDF)
- GPU node types (PDF)

Is there a way to apply for free access?

We offer a pay-for-use service only. Currently, we do not provide free access. However, the policy may change in the future. Please check back with us from time to time.

We are looking for a cloud environment with software installed. Do you provide Software as a Service (SaaS)?

We currently provide a Platform as a Service (PaaS) cloud environment with a few basic HPC software packages installed (Intel compiler, PGI compiler, Intel MPI, OpenMPI, CUDA, and Python). If you need other software packages, you must install them yourself.

What operating system is used in your cloud?

The default operating system is a basic Linux AWS Machine Image (AMI). Others, such as Red Hat Enterprise Linux (RHEL) or or SUSE Linux Enterprise Server (SLES), can be used upon request. However, the non-default operating system will cost more.

How much does it cost?

The cost will be the same as what you get directly from AWS, plus an overhead cost from NASA Enterprise Managed Cloud Computing (EMCC), where we get our cloud resources. There may also be additional overhead from HECC; this is yet to be determined.

Who do I send my NASA Work Breakdown Structure (WBS) funding?

Please send an email to support@nas.nasa.gov. Our User Services manager will contact you to help you transfer the funding to us.

I do not currently have a NAS account. I can bring funding. How do I start?

After HECC receives your NASA Work Breakdown Structure (WBS) funding, we will guide you in obtaining a NAS account. We will then set up your cloud account.

After my cloud environment is created, how do I log in?

Currently, you will use your NAS account to log into a NAS front end with two-factor authentication. From the NAS front end, you will be able to log into your cloud environment.

Do you allow non-NASA projects to use your cloud offering?

We plan to support collaborative work between NASA and non-NASA scientists. However, this effort is still in a planning stage and will not be in production in 2019. Please do share your requirements with us so we can better plan to support them.

How do you plan to allow non-NAS users to login to your cloud?

Currently, the HECC cloud can only be accessed from a NAS front-end system. Non-NAS users will be given NAS accounts (without actual HECC allocations) in order to access a NAS front end. This arrangement may change in the future.

Will I be charged if the cloud resources have problems or do not perform as I expected?

Yes. AWS still charges for usage even if your job fails.

Who do I contact if I need help with issues or job failures on AWS?

NAS staff may be able to help you diagnose some issues, but not all. NASA Enterprise Managed Cloud Computing (EMCC) has a support contract with AWS, and therefore some issues can be examined by AWS engineers. However, it is likely some issues won't be resolved easily or quickly.

AWS Cloud ML/AI

AWS Cloud: Machine Learning/Artificial Intelligence

To support NASA's ever-increasing use of machine learning (ML) tools, we now offer access to the SageMaker environment provided by Amazon Web Services (AWS). With SageMaker and AWS, you can solve your machine learning problems quickly and at a scale not otherwise offered by NAS.

Note: Our current AI/ML environment in the AWS cloud is only rated for low-risk data. Our security plan does not yet allow for moderate-risk or ITAR/EAR99 data in the cloud.

In order to use these tools, you must already have access to the NAS AWS cloud.

For instructions on starting and using these tools, see the following articles:

- Starting a Sagemaker Studio environment
- Starting a Jupyter environment within the Sagemaker environment
- Starting a Jupyter environment within the batch environment associated with your account
- Starting a front end in AWS

Additional AWS ML Services

The following services are also supported by the NAS ML/AI AWS environments. They can be accessed via the AWS GUI console or via the AWS CLI :

- Amazon Translate
- Amazon Polly
- Amazon Transcribe
- Amazon Comprehend

Getting Help

If you need help using or starting a SageMaker/Jupiter environment, contact the NAS support team as follows:

- **Using a SageMaker/Jupyter environment:** send an email to support@nas.nasa.gov requesting help from the ***Data Analytics Team***.
- **Starting a SageMaker/Jupyter environment**: send an email to support@nas.nasa.gov requesting help from the ***Cloud Team***.
  You can also ask the Cloud Team for help with anything related to the AWS batch environment.

Additional Resources

For more information about using the AWS SageMaker graphical user interface (GUI) or integrated development environment (IDE), see the Amazon SageMaker documentation.

Accessing the AWS Management Console Through Cloudtamer

From within the NASA network (or logged into a NASA VPN), log into Cloudtamer at https://cloudtamer.nasa.gov using your NAS AWS credentials (*az_auid*@ndc.nasa.gov) and NASA personal identity verification (PIV) card.

The Cloudtamer dashboard will list your available projectsâ   in most cases, only one project will be listed. In the pulldown menu to the right of the project name, select an AWS account (typically there will be only one option) and the cloud access role within that account (also, typically only one option). Then, select the access type:

- **Web access:** opens a new tab in your browser containing the normal AWS Management Console GUI.
- **Short-term access keys:** opens a pop-up window containing access keys to use with the AWS command-line interface (CLI) tools. These keys are valid for two hours.

Note: Depending on your project and role in that project, there may be only one access type available to select.

Obtaining Short-Term AWS Credentials (Access Keys)

From within the NASA network (or logged into a NASA VPN), log into Cloudtamer at https://cloudtamer.nasa.gov using your NAS AWS credentials (*az_auid*@ndc.nasa.gov) and NASA personal identity verification (PIV) card.

The Cloudtamer dashboard will list your available projectsâ   in most cases, only one project will be listed. In the pulldown menu to the right of the project name, select an AWS account (typically there will be only one option) and the cloud access role within that account (also, typically only one option). Then, select the access type:

- **Web access:** opens a new tab in your browser containing the normal AWS Management Console GUI.
- **Short-term access keys:** opens a pop-up window containing access keys to use with the AWS command-line interface (CLI) tools. These keys are valid for two hours, and are only usable from machines within the NASA network.

Note: Depending on your project and role in that project, there may be only one access type available to select.

To obtain the credentials, hover your cursor over the sections in the pop-up window that contain the credential information, then click to copy to the clipboard. This will make it easy to paste the credentials into an AWS configuration file or to set local environment variables.

Creating a New Front End in AWS

To create a new front end running AWS under your project, you will need to create a new Amazon S3 (Simple Storage Service) object. The S3 object will have the same name as your AWS usernameâ usually, your agency user ID (AUID)â that was set up by NAS staff when your AWS environment was created. The contents of this object determine the type of front end that is created, and how long it will be active.

To create a front end using the simplest format, provide a plain integer value that specifies the number of hours the front end will be active, with a typical maximum of 12 hours. This will start a base-level AWS front end sufficient for file editing, simple pre/post-processing tasks, and limited-bandwidth file transfers.

To create a front end with more specifications, create a file with a line of multiple `name=value` pairs separated by `':'`. The options are:

hour=X
: where X is the number of hours the front end will be active.

mem=X
: where X is the amount of memory, in gigabytes (GB).

arch=type
: where `type` is one of `intel, arm,` or `amd`. The instance will use this architecture.

container=singularity
: enables the use of Singularity containers on the instance.

dcv(=shared)
: enables the Nice DCV GUI front end on the instance. An optional form, `dcv=shared`, will allow all members of the project to log into the front end. Typically, only the user requesting the front end is allowed to access the front end.

jupyter
: starts up a Jupyter notebook running as the user on the front end accessible through SSH forwarding only.

jupyterlab
: Similar to `jupyter`, but will start a Jupyterlab environment running instead of just a Jupyter notebook.

For example, a file containing the following line will start up an Intel-based machine running for 5 hours with at least 64 GB of memory. The Nice DCV GUI environment will be started along with a jupyterlab environment running as the user:

```
hour=5:mem=64:arch=intel:dcv:jupyterlab
```

In comparison, a simple form of a file containing just "5" will start up a basic front-end machine running for 5 hours. The memory and architecture type of a base machine can vary over time, as AWS introduces new instance types. The base instance will be the least expensive processor type that is capable of running all required software, according to NASA security, with at least 1 real CPU core and at least 8 GB of memory available to the user.

Once the local file has been created with the contents corresponding to the type of front end you want to start, it must be uploaded to S3 in either of the following ways:

Uploading the Front-End File Using the GUI

Open a browser window and navigate to the AWS Management Console.

Once the console is open, navigate to the S3 section and select the S3 bucket that has the following format, where *project_name* is the name of your project as assigned by NAS:

```
*-project_name-*-shared
```

Inside this bucket you will see a folder named **remote_frontend_request**. Select this folder. Once you are in the folder, upload the local file you created via the **Upload** button. Again, this **`local file/S3`** object must be named the same as your AUID (username)

Uploading the Front-End File Using the Command Line

Before You Begin: To use this method, you must have the AWS command-line interface (CLI) installed on your local machine. See Installing or updating the latest version of the AWS CLI in the AWS documentation.

With current AWS CLI credentials, the **`aws`** command would look like this:

```
aws s3 cp username s3://s3_bucket_name/remote_frontend_request/
```

The value for *s3_bucket_name* is the name of the shared S3 bucket for your project. Running the command **`aws s3 ls s3://`** will list the buckets in your project's AWS account; the shared bucket will end with **`*-shared`**.

Once the S3 object has been uploaded, it will trigger the process of starting up the new front end. An email will be sent to you with instructions on how to access this front end.

Note: If the **`dcv=shared`** option was used, all users in the project will get the email.

Creating and Accessing AWS SageMaker Notebook Instances

Open a browser window and navigate to the AWS Management Console, as described in Accessing AWS Console Through Cloudtamer.

Once the console is open, navigate to the SageMaker section for the region your project is running in. In the left navigation column, select **Notebook > Notebook Instances**. Any existing notebooks will be shown hereâ   either running or stoppedâ   and, you can also create a new one. To create a new instance, select or specify the following options:

1. **Instance name:** Enter a name that is meaningful to you and other members of the project.
2. **Instance type:** Select the instance type (processor type) you need to run your compute jobs. This can be changed later as your needs change.
3. **Elastic inference:** Enable elastic inference if you know it will help you (check the AWS documentation to learn more).
4. **Additional configuration:** Increase the volume size if you need local (ephemeral) disk space.
5. **IAM role:** Select the Identity and Access Management (IAM) role corresponding to the one with your username in it.
6. **Custom encryption:** You do not need to specify custom encryption.
7. **Network selection:** Only one virtual private cloud (VPC) is listed as an option. Select subnets based on the ability to reach the internet: select any of the **`*-pbs*`** subnets if access to the Internet is needed; otherwise, select a **`*-compute*`** subnet. Note that access *from* the internet is not possible in either subnet.

Creating a Sagemaker Studio IDE

Open a browser window and navigate to the AWS Management Console.

Once the console is open, navigate to the SageMaker section for the region your project is running in. Select **Studio** in the left navigation column to bring up the currently running integrated development environments (IDEs), if any, or the **Create IDE** section.

Select the standard setup, then configure Identity and Access Management (IAM) authentication with the execution role that is listed containing your username. In the configuration setup, select or specify the following options:

- **Virtual private cloud (VPC):** Only one VPC will be listed; select it.
- **Subnets:** Select any `*-compute*` option if you do *not* require Internet access; otherwise, select `any *-PBS*` option
- **Security groups:** Select `*-nas-ssh`
- **Custom encryption:** None is needed.
- **Notebook sharing feature:** Disable, as this feature is not supported.
- **Projects:** Disable.
- **Jumpstart:** Disable.

Using a Jupyter Environment Outside of SageMaker

For simple instance types, you can create a Jupyter environment as described in Creating a New Front End in AWS.

For more complex instance types, such as GPU-based ones, you must start the Jupyter environment from within an existing AWS front end machine. See the **`aws_fe`** command as described in AWS Dynamic Front End.

Note: The article describes how the **`aws_fe`** command is used when on a NAS front end; on an AWS front end, it's the same, except that you do not need to load a module; the **`aws_fe`** command is already in your path.

Modify the various Jupyter kernel options is described in: Using Jupyter Notebook or JupyterLab in the Cloud.

Containers

Singularity

Enabling User Defined Software Stacks with Singularity

Frequent Updates: Support for Singularity is a new offering on NAS systems. The articles in this section are still under development and may be updated frequently as new usage information becomes available.
Singularity containers provide an application virtualization layer that enables both application and environment portability. Using Singularity, you can build a root file system that can run on any Linux system where Singularity is installed, including on NAS systems.

Using Singularity Commands

Basic Singularity commands include:

`singularity help`
Finds information on available options and commands.
`singularity build`
Builds a singularity image.
`singularity shell`
Runs a shell within a container.
`singularity exec`
Runs a command within a container.
`singularity run`
Runs a user-defined default command within a container.

Building a Singularity Container Image

Many ready-to-use Singularity images are available from multiple sources. You can use the **`singularity build`** command to grab an image and use it on your workstation or on Pleiades. For some basic information on using the **`singularity build`** command, see Building an Image Using the Singularity Build Tool.

If you plan to build a Singularity image from scratch using **`singularity build`** or modify an existing one (by using the **`â  â  writable`** option of **`singularity shell`** or **`singularity exec`** commands), root privilege may be required. If you have root privilege on your local workstation, you can install Singularity and build or modify a Singularity image.

On the Pleiades host environment, root privilege is not granted to users. Starting with version 3.8.*x,* Singularity has a **`â  â  fakeroot`** option that allows a user to gain root privilege for the **`singularity build, shell`**, and **`exec`** commands. This option is enabled on NAS systems starting with Singularity version 3.9.8.

Running a Singularity Container Image

If you have an existing Singularity image that is ready to run, you should be able to port and run it on any platform where Singularity is installed. To learn how to run your Singularity container image on Pleiades, see Running a Singularity Container Image on Pleiades.

Pre-Built Machine Learning-Focused Singularity Images

If you would like to use Singularity for machine learning, a few preâ   built images are available in the **`/swbuild/analytix/singularity/images`** directory. For more information about the packages in the images, see Machine Learning: Overview.

References

For more information about Singularity, see the Sylabs Singularity Container Documentation.

Building an Image Using the Singularity Build Tool on Pleiades or Your Local Host

Frequent Updates: Support for Singularity is a new offering on NAS systems. The articles in this section are still under development and may be updated frequently as new usage information becomes available.
This article provides a brief introduction to the **`singularity build`** command, enabling you to build a Singularity image on Pleiades or on your local host. Some information here is also useful if you want to obtain a ready-to-use Singularity image from online resources.

Building a Singularity Image

To build a Singularity image:

```
host% singularity build [local options ...] image_path build_spec
```

where **`image_path`** specifies the output from the build process and **`build_spec`** is a local or remote target. Both are explained below.

image_path (output)

The **`image_path`** output can be one of two formats:

- **Default format:** A compressed, read-only file using the Singularity Image Format (**`*.sif`**). To use this format:

  ```
  host% singularity build ...
  ```
- **Sandbox format:** A directory that is a read-write container. To use this format:

  ```
  host% singularity build --sandbox ...
  ```

Recommendation: It is a common workflow to use the sandbox mode to develop a container where the image is writable, and you can **`shell`** into the image and install software and dependencies (some may require root privilege) until you are satisfied that your container will fulfill your needs. You then build the container as a default Singularity image (**`*.sif`**) for production use. The default format is immutable.

build_spec (target)

A **`build_spec`** target can be local or remote. Acceptable formats are listed below.

Formats for Local Targets

- Definition file: A file that contains a "recipe" for building a container, as described the Singularity documentation.
  A sample definition file is shown in Example 2 below. Get additional github sylabs examples.
- Directory: A directory structure containing a **`(ch)root`** file system
- Image: A local image on your workstation (such as **`*.sif`** or those with legacy Singularity formats)

Uniform Resources Identifiers (URI) for Remote Targets

- **`library://`** (an image library; default: https://cloud.sylabs.io/library)
- **`docker://`** (a Docker registry; default: Docker Hub)
- **`shub://`** (a Singularity registry; default: Singularity Hub)
- **`oras://`** (a supporting Oracle Cloud Infrastructure (OCI) registry)

Examples

Example 1: Build an Image from the sylabs Library

```
host% singularity build lolcow_lib.sif library://sylabs-jms/testing/lolcow
```

Example 2: Build an Image from a Definition File

You may need root privilege to build the image from a definition file. If you do not have root privilege, insert the **`--fakeroot`** option after **`singularity build`** in the commands shown below.

Notes:

- The **`--fakeroot`** option is enabled for use on NAS systems starting with Singularity version 3.9.8.
- The success of building an image from a definition file on your host system is not guaranteed. If it fails, consider building it on a different system or seek assistance from the provider of the definition file.

1. Create a sample definition file (named **`lolcow.def`**) with the following content:

   ```
   BootStrap: library
   From: ubuntu:16.04

   %post
       apt-get -y update
       apt-get -y install fortune cowsay lolcat

   %environment
       export LC_ALL=C
       export PATH=/usr/games:$PATH

   %runscript
       fortune | cowsay | lolcat

   %labels
       Author GodloveD
   ```

   This definition file is copied from the Singularity Quick Start Guide; it successfully builds an image on Pleiades, with the following exception: if **`ubuntu:20.04`** is used in the definition file, it fails to build an image.
2. Build a sandbox directory (named **`lolcow`**) from the definition file:

   ```
   host% singularity build [--fakeroot] --sandbox lolcow lolcow.def
   ```
3. Build a **`.sif`** image (**`lolcow.sif`**) from the sandbox directory:

   ```
   host% singularity build lolcow.sif lolcow
   ```

Running a Singularity Container Image on Pleiades

Frequent Updates: Support for Singularity is a new offering on NAS systems. The articles in this section are still under development and may be updated frequently as new usage information becomes available.
If you have an existing Singularity image that is ready to run, you should be able to port and run it on any platform where Singularity is installed. This article provides information to help you run a Singularity container image on Pleiades.

Note: A container is a runtime instance of an image. You can run multiple containers of the same image.

Unpacking the Image Tar File on Pleiades

If you transferred your Singularity container image to Pleiades as a compressed tar file, use the appropriate tool to uncompress and unpack the tar file. For example:

```
pfe% cd /nobackup/username/dir_containing_the_tar_file
pfe% tar -xzf your_image_name.tgz
```

Accessing Singularity on Pleiades

To find out what versions of Singularity are installed, run:

```
pfe% module avail singularity
singularity/3.x.y    singularity/3.x.z
```

To load the latest version (currently, version 3.9.8), run:

```
pfe% module load singularity
pfe% which singularity
/nasa/singularity/3.9.8/bin/singularity
```

To load a specific version, run:

```
pfe% module load singularity/3.x.z
pfe% which singularity
/nasa/singularity/3.x.z/bin/singularity
```

Running Your Container Image

Depending on what you want to do, you can use one of the following three commands to run the container image. If you include the **`--writable`** option when you run your container, you can also write files within the sandbox directory, provided you have the permissions to do so.

`shell`
: The **`singularity shell`** command allows you to spawn a new Bourne shell within your container and interact with it as though it were a small virtual machine. To exit the shell, enter **`exit`** at the Singularity prompt.

`exec`
: The **`singularity exec`** command allows you to execute a custom command within a container.

`run`
: Singularity containers may contain runscripts. These are user-defined scripts that define the actions a container should perform when it is run. The runscript can be triggered with

the **`singularity run`** command, or simply by calling the container as though it were an executable.

Running Your Container Image: Examples

For example, if you port the lolcow sandbox container to Pleiades, you can run it as described in the following examples.

Notes:

- With Singularity version 3.9.8, you cannot **`shell/exec/run`** the sandbox on your $HOME filesystem. Use it on your Lustre /nobackup filesystem, instead.
- You can also run with a Singularity Image file (**`*.sif`**) instead of a sandbox container. For example, replace the sandbox directory **`lolcow`** in Examples 1-3 with the **`.sif`** file named **`lolcow.sif`**.

Example 1: Using the shell Command

```
pfe% singularity shell lolcow
Singularity> fortune | cowsay | lolcat
 ________________________________________
< You enjoy the company of other people. >
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

Singularity> exit
```

Example 2: Using the exec Command

```
pfe% singularity exec lolcow which cowsay
/usr/games/cowsay
pfe% singularity exec lolcow cowsay hello
 _______
< hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
pfe%
```

Example 3: Using the run Command

```
pfe% singularity run lolcow
 _________________________________
< It's all in the mind, ya know.  >
 ---------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
pfe%
```

Running Graphical Applications in the Container

If your container image includes some graphical applications (for example, xclock or xeyes), you will have to make sure that the DISPLAY environment variable is set inside the container in order to run them. For example,

```
pfe% singularity shell -w your_image_sandbox
Singularity> echo $DISPLAY
ii.jjj.kk.ll:76.0
Singularity> xclock
```

where *ii.jjj.kk.ll* represents the IP address of the specific PFE host. (Its actual value will be different for different hosts.)

Note: Using the **`-e`** option of the **`singularity`** command will clean the environment, including the setting of DISPLAY, before running the container. If the use of **`-e`** is necessary for running your container, you can use the **`--env`** option to pass the DISPLAY variable to the container to run graphical applications.

```
pfe% singularity shell -e --env DISPLAY=$DISPLAY -w your_image_sandbox
```

Note: If you encounter the following warning message, **`cd`** to *your_image_sandbox* and use the **`mkdir`** command to create **`homeX`** before you retry the commands above. (The *X* in **`homeX`** represents the actual value (*X*=1-7) of your $HOME directory on Pleiades.)

```
WARNING: By using --writable, Singularity can't create /homeX destination
automatically without overlay or underlay
```

Setting Up Environment Variables

You can modify environment variables, such as $PATH, within your container. For example:

```
pfe% singularity shell lolcow
Singularity> echo $PATH
/usr/games:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
Singularity> export PATH=$PATH:/homeX/username/bin
Singularity> echo $PATH
/usr/games:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/homeX/username/bin
```

Accessing Files on Pleiades

You can access files on Pleiades from within the container. By default, Singularity bind-mounts $HOME and /tmp into your container at runtime. With recent versions of Singularity, such as version 3.9.8, $PWD inside the container defaults to your Pleiades $HOMEâ *not* to the $PWD on your Pleiades host. For example:

```
pfe% cd /nobackup/username/dir_containing_lolcow_image
pfe% pwd
/nobackup/username/dir_containing_lolcow_image
pfe% singularity shell lolcow
Singularity> pwd
/homeX/username
Singularity> cd /
Singularity> ls
bin   core  environment  home   lib    media  opt   root  sbin         srv  tmp  var
boot  dev   etc          homeX  lib64  mnt    proc  run   singularity  sys  usr
```

You can specify additional directories to bind-mount into your container with the **`--bind`** option. For example, if you want to be able to access files and directories under your **`/nobackuppX/$USER`** directory, you can do:

```
pfe% singularity exec --bind /nobackuppX/username:/mnt lolcow ls /mnt
or
pfe% singularity exec --bind /nobackup/username:/nobackup/username lolcow ls /nobackup/username
```

For read-only filesystems such as **`/nasa`**, optionally add **`:ro`** to the command as follows:

```
pfe% singularity exec --bind /nasa:/nasa:ro lolcow ls /nasa
```

You can bind-mount multiple filesystems. For example:

```
pfe% singularity shell --bind /nasa:/nasa:ro --bind /nobackuppX/username:/mnt lolcow
Singularity> cd /nasa
Singularity> pwd
/nasa
Singularity> cd /mnt
Singularity> pwd
/mnt
Singularity> exit
pfe%
```

Converting Docker Images to Singularity for Use on Pleiades

For security reasons, Docker containers are not supported on Pleiades. If you run Docker containers elsewhere, use one of the methods described below to convert them to Singularity containers for use on Pleiades.

Note: These methods use the **`module load singularity`** command, which will load the latest installed version of Singularity. For more information, see Accessing Singularity on Pleiades.

Method 1: Using an Existing Docker Image on Your Local Machine

1. Find the Docker image ID.

   On the host where you run Docker, use the command **`docker images`** to find the image IDs of Docker images stored in your local registry (usually /var/lib/docker).

   ```
   your_local_system$ docker images
   REPOSITORY        TAG       IMAGE ID        CREATED        SIZE
   hello-world       latest    bf756fb1ae65    5 months ago   13.3kB
   godlovedc/lolcow  latest    577c1fe8e6d8    2 years ago    241MB
   ```
2. Create a tarball of the Docker image.

   For the Docker image you want to port to Pleiades, for example, godlovedc/lolcow with an image ID of 577c1fe8e6d8, create a tarball using the **`docker save`** command:

   ```
   your_local_system$ docker save 577c1fe8e6d8 -o lolcow.tar
   ```
3. Copy the tarball to Pleiades.

   Use **`scp`** or **`sup shiftc`** (see the article Using Shift for Remote Transfers and Tar Operations) to copy to Pleiades. For example:

   ```
   your_local_system$ scp lolcow.tar pfe:
   ```
4. Convert the tarball to a Singularity image.

   On a Pleiades front-end system (PFE), load the singularity modulefile prior to running any Singularity commands:

   ```
   pfe$ module load singularity
   ```

   If the tarball is in your current working directory on Pleiades, do:

   ```
   pfe$ singularity build --sandbox lolcow docker-archive://lolcow.tar
   ```

   If the tarball is not in the current working directory, specify the path, for example, /tmp:

   ```
   pfe$ singularity build --sandbox lolcow docker-archive:///tmp/lolcow.tar
   ```

   Note: In this example, lolcow is the directory name of the Singularity image.
5. Run the Singularity sandbox as usual. For example:

   ```
   pfe$ singularity shell lolcow
   pfe$ singularity exec lolcow cowsay hello
   pfe$ singularity run lolcow
   ```

Method 2: Using an Existing Docker Image on Docker Hub

Use this method if your ready-to-use Docker image has been uploaded to the Docker hub. For example, the image godlovedc/lolcow exists in the Docker hub.

Note: You need an account to upload to the Docker hub.

On a Pleiades front-end system (PFE), do the following to create a singularity image or a singularity sandbox:

```
pfe$ module load singularity
pfe% singularity pull lolcow.sif docker://godlovedc/lolcow
# or
pfe$ singularity build --sandbox lolcow docker://godlovedc/lolcow
```

Method 3: Using a Working Dockerfile Without a Docker Image

Use this method if you do not have a ready-to-use Docker image but you have a working Dockerfile (a text file containing all commands needed to build an image).

1. Using the Dockerfile as a guide, create a Singularity definition file. See Example 2 in the article Building an Image Using the Singularity Build Tool for a sample definition file.
2. Build a Singularity image from the definition file.

   ```
   pfe$ module load singularity
   pfe$ singularity build [--fakeroot] --sandbox lolcow lolcow.def
   ```

For assistance with converting a Dockerfile to a Singularity definition file, contact support@nas.nasa.gov.

Running Singularity in Hybrid MPI Mode

Running Singularity in hybrid MPI mode allows you to call **`mpiexec`** or a similar launcher on the **`singularity`** command itself, when you execute a Singularity container with MPI code. The MPI process outside of the container will then work in tandem with MPI inside the container and the containerized MPI code to instantiate the job.

Singularity supports two main open-source implementations of MPI: OpenMPI and MPICH. This article provides information primarily on using OpenMPI to run on NAS systems.

How OpenMPI Interacts with Singularity in Hybrid Mode

The workflow described on the Singularity website is duplicated here:

1. The MPI launcher (e.g., **`mpirun, mpiexec`**) is called by the resource manager or the user directly from a shell.
2. OpenMPI then calls the process management daemon (ORTED).
3. The ORTED process launches the Singularity container requested by the launcher command.
4. Singularity instantiates the container and namespace environment.
5. Singularity then launches the MPI application within the container.
6. The MPI application launches and loads the OpenMPI libraries.
7. The OpenMPI libraries connect back to the ORTED process via the Process Management Interface (PMI).

At this point, the processes within the container run as they would normally, directly on the host.

The MPI in the container must be compatible with the version of MPI available on the host. If performance is critical, the MPI implementation in the container must be configured for optimal use of the hardware. Since the MPI implementation in the container must be compatible with the version available on the system, a standard approach is to build your own MPI container, including the target MPI implementation.

Build Your Own OpenMPI on the Host

It is likely that there is no compatible OpenMPI installation for your container on NAS systems. To build a compatible OpenMPI on the host, check the version and configuration inside the container first. You can find this information by running the **`ompi_info`** command. The following example shows OpenMPI version and configure options used inside a container.

```
pfe% singularity shell your_container_sandbox
Singularity> ompi_info
                Open MPI: 3.1.4
...
  Configure command line: '--enable-orterun-prefix-by-default' '--with-verbs'
                          '--prefix=/usr/local/mpi'
...
```

To build your own OpenMPI on the host:

1. Download an OpenMPI software version that is the same as, or newer than, the version used inside the container.
2. Install the software in your /nobackup filesystem by using **`--prefix=/nobackuppX/username/...`**

   Note: NAS users do not have permission to install OpenMPI in any system location, including the /usr/local directory.
3. Configure the OpenMPI with options similar to those used inside the container.

The option **`--with-verbs`** is used to build support for OpenFabrics verbs (previously known as "Open IB"). Since OpenMPI will try to build support for every feature that it can find on a system by default, and NAS systems have an InfiniBand (IB) network, you do not need to include this option to produce an OpenMPI installation that supports IB.

Note: If the option **`--with-pmi`** (where PMI means Process Management Interface) is included as a configure option inside the container, do *not* include it when you configure your OpenMPI on the NAS host. This is because PMI supports launching jobs with the SLURM workload manager rather than PBS, which is used at NAS.

Test the Hybrid MPI Mode with a Simple MPI Code

Before running your MPI application in hybrid mode, compile a HelloWorldMPI code using the compiler and MPI library inside the container and run it in hybrid mode. If the HelloWorldMPI code does not run properly, your MPI application won't either.

Note: If the **`singularity exec -e`** option is used, it likely would result in every MPI rank acting as rank 0, as demonstrated in the following example, which uses a total of four MPI processes (from two nodes with two MPI processes on each node):

```
mpirun -np 4 -H r101i3n12:2,r101i3n13:2 \
singularity exec -e your_container_sandbox $HOME/hello_world_mpi_built_inside_container

Hello world from processor r101i3n12, rank 0 out of 1 processors
Hello world from processor r101i3n13, rank 0 out of 1 processors
Hello world from processor r101i3n12, rank 0 out of 1 processors
Hello world from processor r101i3n13, rank 0 out of 1 processors
```

Depending on the required setup (such as removing the **`-e`** option and adding some MPI-related environment variables), a successful run with HelloWorldMPI should result in:

```
Hello world from processor r101i3n12, rank 0 out of 4 processors
Hello world from processor r101i3n13, rank 2 out of 4 processors
Hello world from processor r101i3n12, rank 1 out of 4 processors
Hello world from processor r101i3n13, rank 3 out of 4 processors
```

Run the Application with OpenMPI in Hybrid Mode

The following is a more elaborate example of how to run applications built with OpenMPI in hybrid MPI mode. A total of four MPI processes is used (from two compute nodes with two MPI processes on each node). The OpenMPI inside the container is installed in the /usr/local/mpi directory.

```
mpiexec -np 4 -H r101i3n12:2,r101i3n13:2 --mca btl vader,openib,self \
singularity exec -B /nasa:/nasa -B /nobackuppX/username:/nobackuppX/username \
--env OPAL_PREFIX="/usr/local/mpi" --env PMIX_INSTALL_PREFIX="/usr/local/mpi" \
your_container_sandbox a.out
```

- The **`-H`** option is used to properly list the compute nodes to invoke processes on (you will need to find the hostnames of these nodes, such as r101i3n12 and r101i3n13). Without the **`-H`** option, all of the processes would run on the first node, possibly oversubscribing the node.
- If the OpenMPI libraries on the host and in the container were built to support **`openib`**, use **`â  â  mca btl vader,openib,self`** to explicitly use the **`vader, openib, and self`** byte transfer

layer (BTL) components of the modular component architecture (MCA) to transfer data between two processes.

Read the next section to learn more about the BTL components.

- The **`-B`** option is used to bind mount the Pleiades /nobackup filesystem and the /nasa directory, which is necessary if any required software is installed in those locations (such as GCC and OpenMPI on the host).
- **`singularity exec --env`** is used to properly set the environment variables OPAL_PREFIX and PMIX_INSTALL_PREFIX to the MPI installation inside the container and ensure they are seen by every MPI process on every node. This may be needed to resolve conflicts between the host and the container. In fact, there is more than one way to achieve this effect. For example, you can add the options **`â  x OPAL_PREFIX â  x PMIX_INSTALL_PREFIX`** to **`mpiexec`**, with all of them set to the /usr/local/mpi directory inside the container:

  ```
  mpiexec -np 4 -x OPAL_PREFIX=/usr/local/mpi -x PMIX_INSTALL_PREFIX=/usr/local/mpi ...
  ```

  Note: On the host environment, the OPAL_PREFIX and PMIX_INSTALL_PREFIX variables still need to be set to the MPI installation on the host in order for **`mpiexec`** to work properly.

More Information About the OpenMPI MCA BTL

You can see which BTL components are supported in your OpenMPI installation by running the **`ompi_info`** command. For example:

```
Singularity> ompi_info | grep 'MCA btl'
             MCA btl: self (MCA v2.1.0, API v3.0.0, Component v3.1.4)
             MCA btl: openib (MCA v2.1.0, API v3.0.0, Component v3.1.4)
             MCA btl: tcp (MCA v2.1.0, API v3.0.0, Component v3.1.4)
             MCA btl: vader (MCA v2.1.0, API v3.0.0, Component v3.1.4)
```

Among these components, **`self`** is used for loopback communication (i.e., when an MPI process sends to itself); **`openib`** uses the OpenFabrics Alliance's verbs to support InfiniBand, RDMA over Converged Ethernet (RoCE), and RDMA over Internet Protocol networks (iWARP); **`tcp`** uses TCP-based communications over IP interfaces and networks; and **`vader`** uses shared memory.

It is usually unnecessary to specify the BTL components on the **`mpirun`** command line, as OpenMPI will use the best BTL available for each communication. They are typically used only when you want to be sure to use the specific BTL, or when **`^openib`** (which means "no openib") is set as the default in your OpenMPI modulefile and you want to be sure that **`openib`** is used, in order to avoid slow communication and possible time-outs due to the use of **`tcp`** for some container applications.

Note: If you are specifying BTL components explicitly, be sure to include **`self`**. If the **`self`** component is not specified, OpenMPI may be unable to complete send-to-self scenarios.

Check if You are Using a NAS-Instrumented mpirun or mpiexec

If you are using an NAS-built MPI modulefile on the host, it is likely that **`mpirun`** and **`mpiexec`** have been instrumented. The non-instrumented **`mpirun`** and **`mpiexec`** commands are renamed to **`mpirun.real`** and **`mpiexec.real`**.

If the instrumented **`mpirun`** and **`mpiexec`** on the host fail to run the container, try using **`mpirun.real`** or **`mpiexec.real`** instead.

TIP: Many of the containers (and their usage instructions) that you find online are meant for running with the SLURM workload manager. You may have to experiment and modify the usage to work with PBS.

References

- Singularity and MPI applications
- OpenMPI Frequently Asked Questions

Best Practices for Running Singularity on NAS Systems

Singularity containers provide an application virtualization layer that enables both application and environment portability. Using Singularity, you can build a root file system that can run on any Linux system where Singularity is installed. This article provides best practices for using Singularity on NAS systems.

Do Not Pull a Singularity Image to your Pleiades Home Directory

Singularity images can be many gigabites (GB) in size. Avoid pulling images into your Pleiades home directory, which by default has a soft quota limit of 8 GB. The recommended methods for pulling images are:

```
pfe% cd /nobackup/username
pfe% singularity pull your_container.sif source_of_your_container
or
pfe% singularity build your_container.sif source_of_your_container
```

Note: Singularity creates a **`.singularity`** directory in your Pleiades home directory. For some images, if you use simply **`singularity pull`** ***`source_of_your_container`*** (without specifying the .sif file), it would place the *your_container*.sif file in the **`.singularity`** directory, which may consume much of your home space quota.

Running a Sandbox May Be More Efficient than Running the .sif File

During runtime, the .sif file will be converted to a sandbox behind the scenes before being started. The conversion from .sif to sandbox may take a long time or even hang, especially if you are running the container in parallel where there will be multiple conversions at the same time. Consider converting the .sif file into a sandbox explicitly ahead of time, and running the container with the sandbox instead of the .sif file in your run script:

```
singularity build --sandbox your_container_sandbox your_container.sif
singularity exec [options] your_container_sandbox
```

You May Need to Create a /home*X* Directory in your Container Sandbox

We have seen the following error message in Singularity version 3.6.4 or later, but not version 3.5.3:

```
pfe% singularity shell -w your_container_sandbox
FATAL: container creation failed: mount .../homeX -> /homeX
error: while mounting ..../homeX: destination /homeX doesn't exist in container
```

where **`home`*`X`*** is the Pleiades home filesystem you are assigned to use. You can resolve this error by creating a /home*X* directory in the container sandbox:

```
pfe% cd /path_to_your_container/your_container_sandbox
pfe% mkdir ./homeX
```

Ensure All Host Filesystems Needed for Read and/or Write are Mounted

When Singularity swaps the host operating system for the one inside your container, the host filesystems become inaccessible. If you need access to any of your host filesystems or directories (such as /nasa for a certain module or your /nobackup for reading/writing data) you must bind-mount them with the **`-B`** or **`--bind`** command-line option, or by setting the SINGULARITY_BIND environment variable:

```
singularity shell -B /nasa:/nasa \
-B /nobackuppX/username:/nobackuppX/username your_container_sandbox
```

where **`/nobackuppX`** is your assigned /nobackup directory.

Note: Not mounting required filesystems is one of the most common failures we have experienced when running Singularity containers.

Know the Environment Settings of your Container and the Effect of Using the `-e` Option

Environment variables set in your current session after sourcing your system startup file, such as .cshrc or .bashrc, or after loading a modulefile, are exported into the container. The only exceptions are PATH and LD_LIBRARY_PATH, which are set differently inside the container compared to the host environment.

Do not load unnecessary modulefiles on the host before running your container. The variables set in these modulefiles may confuse the environment inside the container.

The **`-e`** or **`--cleanenv`** option of the **`singularity [run, exec, or shell]`** command lets you clean the environment before running the container. As shown in the table below, adding **`-e`** to the **`singularity exec`** command retains variables on the left column but removes those on the right column for a openform7-paraview56 container. Among the retained variables, $PATH and $LD_LIBRARY_PATH are set differently between the host and container environments.

Variables retained with -e enabled	Variables removed with -e enabled
SINGULARITY_XXX	DISPLAY, PYTHONSTRTUP
PATH (re-defined), LD_LIBRARY_PATH (re-defined)	SSH_XXX, XXX_RSH, SYSTEMD_LESS
PWD, HOME, uid, user	xxxMODULExxx, MANPATH, LMFILES
LANG, TERM, SHELL, SHLVL, PS1	HOSTxxx, MACHTYPE, OSTYPE, VENDOR
.	USER, USER_PATH, LOGNAME, GROUP
.	MAIL, EXINIT, CSHEDIT, OSCAR_HOME

There may be a difference in behavior when running your container with or without the **`-e`** option. For example, if you want to run a graphical application, such as ParaView, within the container, you should add **`--env DISPLAY=$DISPLAY`** to the **`singularity shell`** command if the **`-e`** option is used to start the container:

```
singularity shell -e --env DISPLAY=$DISPLAY your_container_sandbox
```

Likely Places to Find Executables or Libraries Inside the Container

For many containers, the default PATH or LD_LIBRARY_PATH might not include paths of executables or libraries needed to run the application of interest. If the container includes MPI libraries, they are commonly installed inside the container in the /usr/local or /usr/local/mpi directories. Other packages are likely installed under the /opt directory of the container. If you still cannot find the executables or libraries and there is no documentation about the container, contact the provider of the container.

Running a Singularity Container Image in AWS

In our Amazon Web Services (AWS) environment, Singularity is built into the base compute image (operating system), so you do not need to load a module in order to run a Singularity container in AWS. Normal operation is nearly identical to the Pleiades environment described in Running a Singularity Container Image on Pleiades, with one main difference described below.

Building/Converting SIF Files in AWS

If you want to build your own SIF files, or convert an SIF file into a sandbox image in the AWS environment, you must add the **`container=singularity`** option to the typical PBS **`â  l select`** command line, as follows:

```
#PBS â  l select=2:mpiprocs=16:container=singularity:mem=64
```

This option increases the size of the /tmp filesystem and points the Singularity cache and TMPDIR environment variables to this filesystem. This is helpful both when you build SIF images and when you run them.

Note: When you start a front end in AWS using the **`aws_fe`** command, the **`â  â  container singularity`** command line argument provides the same functionality as the **`container=singularity`** does in the **`â  l select`** line.

For more information, see AWS Dynamic Front End.